Optimization Services (OS)

[Summary for Review by the George B. Dantzig Dissertation Award Committee]

Jun Ma

NORTHWESTERN UNIVERSITY

EVANSTON, ILLINOIS

JUNE, 2005

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

Committee

Robert Fourer, Northwestern University

Chairperson of Ph.D. Committee and Advisor

Kipp Martin, University of Chicago

Co-advisor

John R. Birge, University of Chicago

Wei Chen, Northwestern University

Sanjay Mehrotra, Northwestern University

Thomas Tirpak, Motorola Inc.

Table of Contents

1. INTRODUCTION TO OPTIMIZATION SERVICES

Optimization Services is a unified *framework* for the next generation distributed optimization systems, mainly optimization over the Internet. The corresponding Optimization Services Protocol is being promoted as a set of industrial *standards*. The phrase "next generation" emphasizes the fact that Optimization Services is a state-of-the-art design and is *not* adapted from any existing system. It also suggests that the OS framework fits well in the general picture of the "Future of Computing."

1.1 Future of Computing and the Optimization Services Motivations

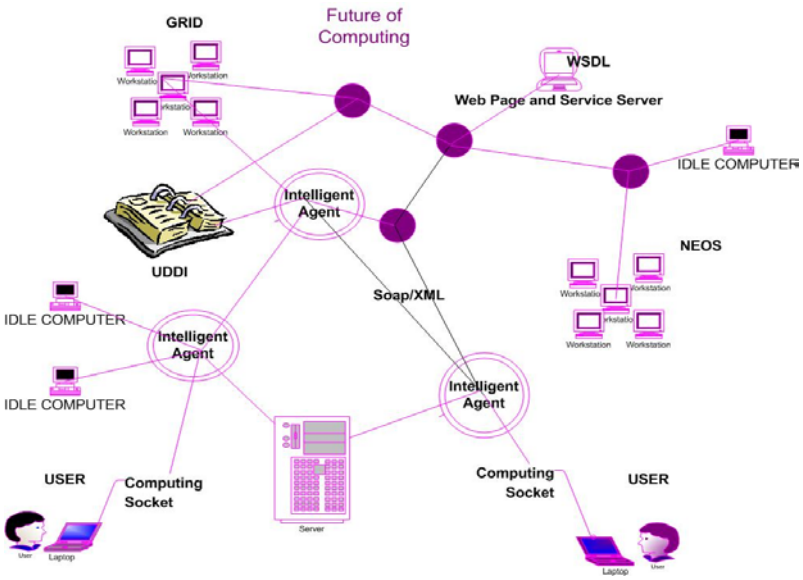

Figure 1-1: Future of computing.

Figure 1-1 depicts a future computing framework in which semantic Web services and software agents interact with each other. A "consumer" plugs his computer into a so-called "computing socket" (or a wireless access point), which is presumably next to the electrical and phone outlets. Computing then is solely viewed as part of the daily utilities that are ubiquitously available (thus the coined name Optimization Services). The consumer does not need to know which computer or grid of computers his requested services are finally run, just as he does not need to know where his electric power is generated or where the water flows in from.

Although many of the OR related tasks can be done by a combination of manual labor and custom tools using existing OR technologies, the OR community needs a combination of software and systems with standardized interfaces, seamless integration, embedded intelligence, and no human involvement. Our research in Optimization Services is motivated by the fact that although large-scale optimization has been a subject of research for over half a century now, the challenge of making it useful in practice is still a problem. Initially the greatest difficulties were posed by solution computation and model building, but the primary impediment to broader use of optimization models and methods today is one of communication.

Currently there are many optimization solvers, various formats to represent optimization problems, and heterogeneous mechanisms to communicate with optimization components. There are also numerous research initiatives in developing supporting tools to analyze and benchmark optimization problems and solvers. Moreover, different optimization components are implemented in different programming and modeling languages and located on different platforms locally or all over the network. Even if a prospective user is not puzzled by such a plethora of combinations, the trouble of obtaining, installing and configuring the OR software does not justify the benefits from using it.

1.2 Overview of Optimization Services (OS)

In the early history of solving the mathematical programs, the translation of an optimization model to a format required by a linear program solver involved intensive human labor and human labor alone. The first major attempt to provide an environment to help the solution of a mathematical program was the matrix generator. A matrix generator is a computer code that creates input in the form of coefficient matrices for a linear solver. The task of translation from the modeler's form to the algorithmic code's form is thus divided and shared between human and computer. The dominance of matrix generator continued to the early 1980's.

Then there was a big breakthrough with the development of modeling languages (the first major one being GAMS), which entirely shifted the human labor of translation to computer. In 1983, Robert Fourer articulated a contrast between the modeler's view and the algorithm's view. He described new design considerations that would combine strength of general, high level languages with special-purpose languages [3]. Modeling languages introduced two key ideas: separation of the data from the model and separation of modeling language from the solver. They addressed the issues of verifiability, modifiability, documentability, independence, simplicity, and other special drawbacks of matrix generators. As modeling languages began to be packaged with other auxiliary tools that assist in model construction, people started to call them modeling systems.

It has become increasingly common to separate modeling languages and systems from optimization solvers. In fact, the modeling language software, solver software, and data used to generate the model instance might reside on different machines using different operating systems. The next great leap forward happened in the mid 1990's when large-scale optimization was brought onto the Internet. The NEOS Server [2] for Optimization is the most ambitious realization to date of the optimization *server* idea. A cooperative effort of over 40 designers, developers, collaborators, and administrators at the Optimization Technology Center of Northwestern University and Argonne National Laboratory, NEOS provides access to dozens of solvers. Modelers can submit problems with representations of many kinds and through networking mechanisms based on nearly all major protocols.

By using distributed computing technologies such as XML and Web services, we envision the Optimization Services approach as the next step in the evolution of optimization technologies. Optimization Services is an XML-based, service-oriented, optimization centered, distributed and decentralized architecture. By using Optimization Services Protocols, Optimization Services enable OR software to integrate with partners and clients in a fashion that is loosely coupled, simple, and platform-independent.

1.2.1 OS as a framework for optimization systems

Optimization Services is a framework that specifies how a set of cooperative classes and interfaces should be designed and implemented in order to solve an optimization problem. The Optimization Services framework has the following properties:

- It consists of multiple classes or components, each of which may provide an abstraction of some particular optimization concept.
- It defines how these abstractions work together to solve an optimization problem.
- Its optimization-related components are reusable, which is what makes Optimization Services a good framework, since it provides generic behavior that many different types of OR applications can use.
- It organizes patterns at a higher level. By "pattern" we mean a tried and true way to deal with an optimization process, from the whole context to the problem and to the final solution that appears over and over again. Thus the adopted patterns in the Optimization Services is an effective means of communication between OR software components, therefore bringing order into chaos.

There is a key difference between a library and a framework. A library contains functions or routines that an application or a user can invoke. A framework provides generic, cooperative components that software can follow and extend. Figure 1-2 shows the difference between a framework and a library. The Optimization Services framework provides a foundation upon which OR applications, software, and libraries are built, whereas an OR library is a piece of software used by other OR applications.

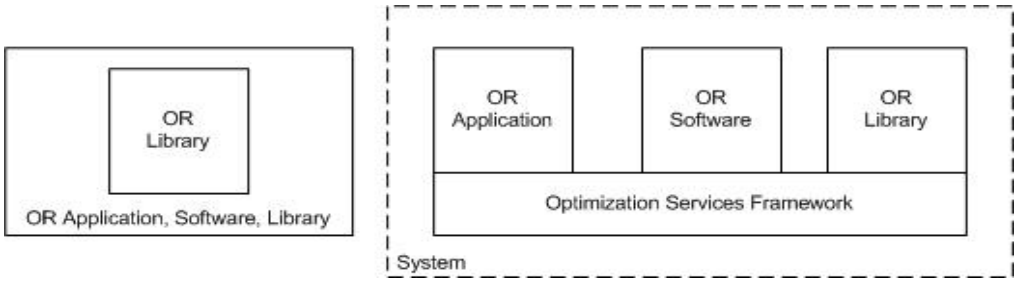

Figure 1-2: Difference between an OR library and the Optimization Services framework.

Although Optimization Services is intended to be a standard framework, *not* a system, we are also developing the optimization system according to this framework (see http://www.optimizationservices.org [3] or http://www.optimizationservices.net [9]) and building libraries for other people to more easily put up their OS software and components.

1.2.2 OS as a computational infrastructure for Operations Research (OR)

Operations Research, as a branch of applied mathematics, has its foundations in mathematics, computing and economic theories, on which basic tools in optimization and simulation are built. We apply these tools to model problems in such areas as manufacturing, distribution, finance, and marketing.

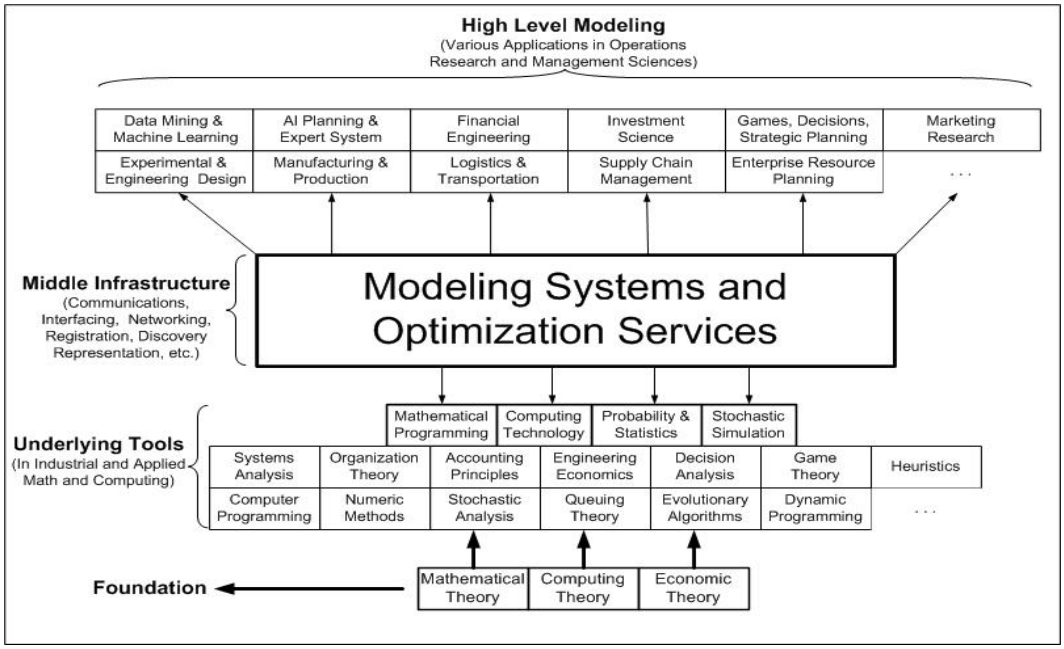

Figure 1-3: Positioning of OS in the hierarchy of Operations Research (OR).

Figure 1-3 shows a hierarchy of operations research activities. The highest level in the hierarchy is concerned with modeling and is the part that directly interfaces to consumers who use models for daily analysis. The level of "Underlying Tools" comprises such core areas as mathematical programming, simulation, and statistics. This level is typically regarded as what uniquely defines Operations Research.

Optimization Services' position is in the middle of the Operations Research hierarchy. It is concerned with things like communication infrastructures, modeling languages and systems. It is an interface part that bridges OR modeling with OR tools. When implemented smoothly, it is the part that is not noticed by modelers or users.

1.2.3 OS as the next generation Network Enabled Optimization System (NEOS)

Figure 1-4: NEOS Server for Optimization at http://www-neos.mcs.anl.gov.

The NEOS server of the Optimization Technology Center of Northwestern University and Argonne National Laboratory makes more than 50 solvers available through several network mechanisms. Because

the Server has evolved along with the Web and the Internet from their early times, it is limited to some degree by initial design decisions and is facing growing communication difficulties.

Optimization Services, with all the OR applications, software and libraries built upon the OS framework, is intended to be the next-generation NEOS. It addresses many outstanding design and implementation challenges faced by the current NEOS under the large-scale and distributed optimization environment. For example, the benefit to the optimization community of a common format for instance representation and an accepted application programming interface (API) for solvers is clear. If modeling languages support a common format (Optimization Services instance Language– OSiL, §2.4), and solvers support a common API that operates on the instance format, then solver developers do not have to worry about supporting multiple model formats and modeling language developers do not have to worry about supporting varied solver input formats. Using the standard representation of an instance, only M + N (instead of M × N under the current status) drivers are required for complete interoperability (Figure 1-5).

As stated in the original National Science Foundation (NSF) proposal [4] for this research, titled *Next-Generation Servers for Optimization as an Internet Resource*:

"The planned research is motivated by a vision of a next-generation NEOS Server that addresses outstanding challenges of communication in large-scale optimization. This work will address design as well as implementation issues posed by standardizing problem representations, automating problem analysis and solver choice, working with new web-service standards, scheduling computational resources, benchmarking solvers, and verification of results — all in the context of the special requirements of large-scale computational optimization."

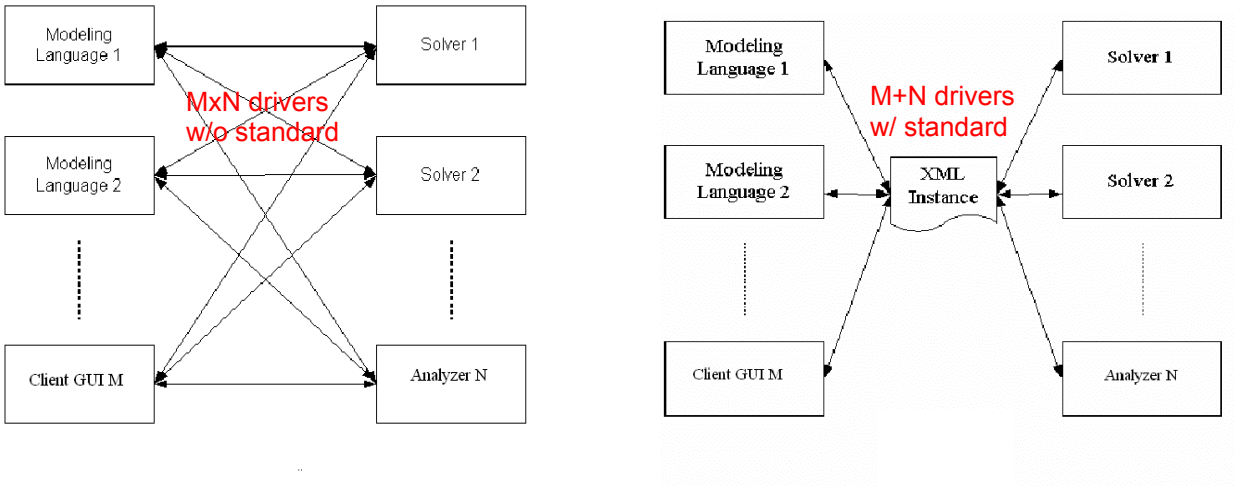

Figure 1-5: M+N drivers needed by M modeling languages (or GUIs) and N solvers (or analyzers) with a standard XML instance.

Considering the fact that the NEOS Server has over the past decade shown significant value in helping users of all kinds, Optimization Services can have widespread benefits to practitioners inside and outside of

the Operations Research community. The continuing goal of Optimization Services as the next generation NEOS should stay the same as the current NEOS, to "make optimization a part of the worldwide software infrastructure that supports science and commerce."

1.2.4 OS as the Operations Research (OR) Internet

There is one fundamental difference between the current NEOS and Optimization Services. NEOS is based on a tightly coupled centralized structure. All the solvers are connected with the server, and all the optimization job requests have to go through it. Therefore, the system does not scale well.

On the other hand, Optimization Services adopts a decentralized Service-oriented Architecture. There is still in some sense a "central" server in the middle, but it functions as a lightweight "registry server," or just "registry." Such a registry knows all the solvers and other Operations Research software that exist in the whole decentralized system by keeping metadata files. Metadata here means that the registry contains information *about* the software, but not the software itself. No solvers are actually executed by this registry; instead users directly contact the solvers in a peer-to-peer mode. The advantages of a decentralized Service-oriented Architecture are significant and are elaborated throughout this thesis. The Internet has become popular because it is a decentralized architecture. There is no such thing as a "central repository server" that hosts all the Web pages. Development and maintenance all happen spontaneously. It is our vision that a decentralized architecture can better promote research and development in Operations Research. In Section 2, we discuss the Optimization Services process, which illustrate the close analogy between the Optimization Services and the Internet architectures.

1.3 Optimization Services Protocol (OSP)

A protocol is an agreed upon format for transmitting data between two devices, hardware or software. The Optimization Services Protocol determines how optimization related data are *represented* and *communicated* between two Optimization Services compatible software components.

OSP is a rapidly evolving set of standards that consists of over 20 sub-protocols, all described by an abbreviation of in the form of "OSxL", meaning some Optimization Services x Language. For example, OSiL stands for Optimization Services instance Language, which is a language expressed in XML to specify the structure and format of general optimization instances. As a core of the Optimization Services framework, OSP has great promise for the world of Operations Research applications, optimization systems and distributed computing.

1.3.1 OSP as an application level protocol in protocol layering

In modern protocol design, protocols are "layered." Layering is a design principle that divides the protocol design into a number of smaller parts, each of which accomplishes a particular sub-task, and interacts with the other parts of the protocol only in a small number of well-defined ways. For example, one

layer might describe how to encode text (with ASCII, say), while another may detect and retry errors (with TCP, the Internet's Transmission control protocol), another handles addressing (with IP, the Internet Protocol). Layering allows the parts of a protocol to be designed and tested without a combinatorial explosion of cases, keeping each design relatively simple.

As illustrated in Figure 1-6, the *reference* model usually used for layering is the Open Systems Interconnection (OSI) seven layer model -- physical, link, network, transport, session, presentation, and application layers from bottom to top. The Internet protocols (TCP/IP) can be analyzed using the OSI model, even though TCP/IP has only four distinct layers -- network access (e.g. Ethernet), internet (e.g. IP), transport (e.g. TCP), and application layers (e.g. HTTP). All protocols layered above the HTTP protocol (e.g. SOAP [14], briefly described in the next section) are also called application level protocols. Thus OSP, being a protocol above SOAP, is classified as an application level protocol. As a result, no mechanisms such as encoding and security are addressed in OSP. OSP leverages on the mechanisms provided by its underlying protocols, for example the encoding scheme from SOAP and the security support from HTTP.

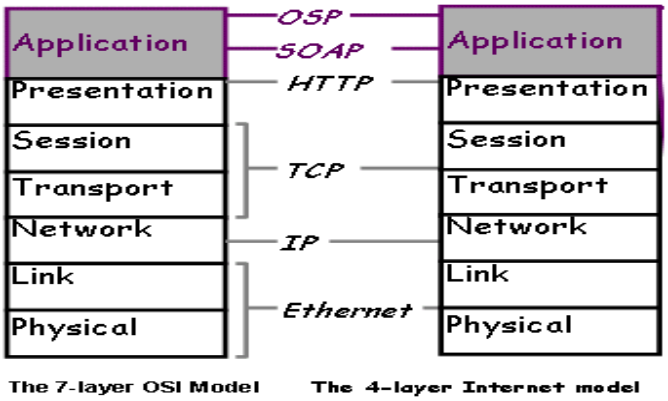

Figure 1-6: Layering of Internet protocols.

1.3.2 OSP as an interdisciplinary protocol between CS and OR

The Optimization Services Protocol is entirely based on SOAP[1]. Short for Simple Object Access Protocol, SOAP is a lightweight XML-based messaging protocol used to encode the information in Web service request and response messages. SOAP messages are independent of any operating system or protocol and may be transported using a variety of Internet protocols, including SMTP, MIME, and HTTP, although nearly always it is using HTTP. Generally, the protocols under the network layer belong to the area of Electrical Engineering, and the protocols above the network layer belong to the area of Computer Science. In this regard, SOAP is naturally a Computer Science protocol.

Although SOAP defines a set of rules for *structuring* messages, it does not specify the actual *content* of the messages. In that sense SOAP is a generic and domain-independent protocol. OSP takes on the task of

[1] More exactly, it is our implementation of the Optimization Services Protocol (OSP) that is entirely based on SOAP. OSP can be built on other similar protocol, but the XML nature of OSP and SOAP make them a natural pair.

specification of the content in the domain area of Operations Research. The nature of bridging protocols in two separate areas, Computer Science and Operations Research, classifies OSP as an interdisciplinary protocol. In an actual data packet, all the contents specified in OSP are inside a SOAP envelope. As both OSP and SOAP are XML based protocols, this is equivalent to saying that OSP contents are child elements of a SOAP parent element (Figure 1-7). For example, the Optimization Services hookup Language (OShL) sub-protocol of OSP specifies that OS compatible solvers should provide an invocation in the form:

```
String solve(String instance);
```

in which the input string "`instance`" has to follow the representation format specified by the Optimization Services instance Language (OSiL) and the output string has to follow the representation format specified by the Optimization Services result Language (OSrL).

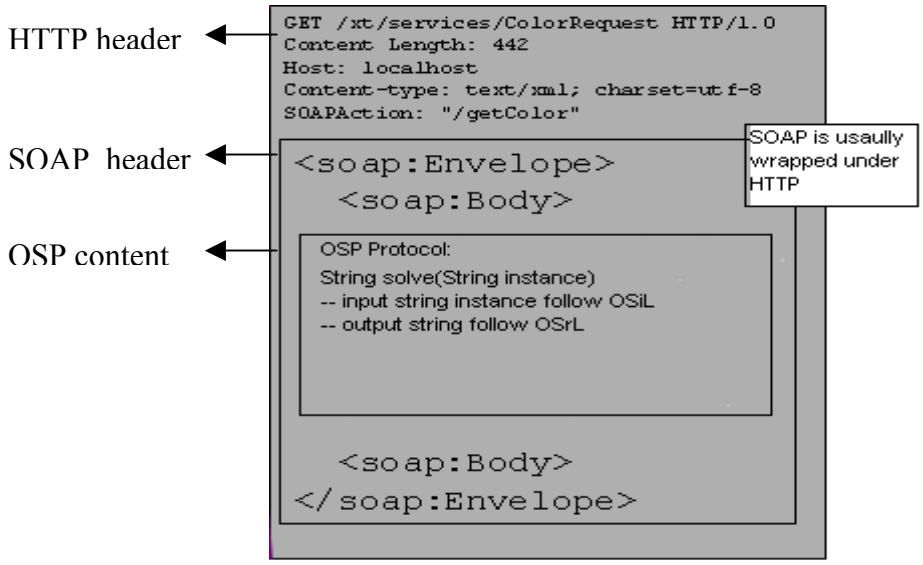

Figure 1-7: OSP inside SOAP, which, in turn, is usually inside HTTP.

1.3.3 OSP sub-protocols

There are mainly two categories of OSP sub-protocols, one that deals with representation (§2.4) and the other that deals with communication (§2.5). All the sub-protocols described in the OS Registry section (§2.6), actually belong to either the representation or communication category. Since the registry is one of the most significant parts of Optimization Services and there are numerous corresponding sub-protocols, all the registry related sub-protocols are listed in a separate section (§2.6),. Most of the representation sub-protocols are specified in XML schema, a mechanism for defining a vocabulary specifying the structure of XML documents [12][13]. Most of the communication sub-protocols are specified in Web Services Description Language [15], a mechanism to describe the invocation syntax of a Web service, e.g. an optimization service.

2. OPTIMIZATION SERVICES FRAMEWORK

2.1 Optimization System Components

First we clarify certain terminology usage in this thesis. Most modeling language software starts with a *core* modeling language along with a language compiler. Gradually the core evolves to include other *auxiliary* software such as preprocessors and graphical user interfaces (GUIs). By "auxiliary" we mean tools that *help* in constructing, preprocessing and compiling a modeling language, but *not* solving the model, which is the function of a solver. Modeling languages are eventually packaged with solvers in distribution. The whole package is usually called a modeling system. In this thesis, however, we stay away from using the term "modeling system" to avoid its potential confusion with the more general optimization system (Figure 2-1). Instead we call a modeling language without any solvers a Modeling Language Environment (MLE), i.e., a modeling language core with only auxiliary tools. An MLE is a component in an optimization system.

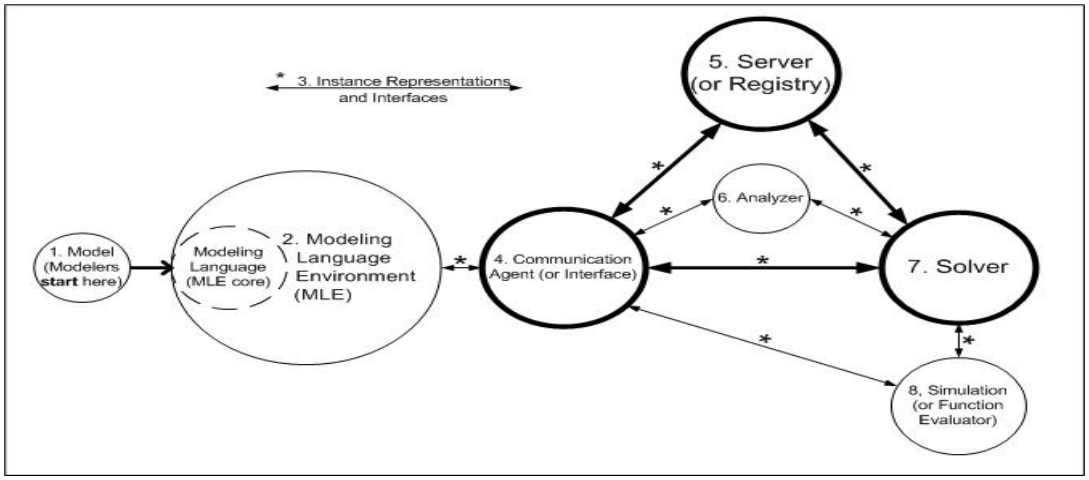

Figure 2-1: A typical optimization system and component interaction.

An optimization system contains most of the following components:

1. Model. This is where a modeler starts. The model component differs from the rest of the optimization components in that it is an abstraction of an input problem rather than a physical piece.
2. Modeling Language Environment (MLE). The core of the MLE is the modeling language, in which an abstract model is defined. The MLE helps in the implementation process. Often a modeling environment may not have a modeling language, but just a spreadsheet or some graphical user interfaces with implicitly defined models. We call it a GUI. From the perspective of Optimization Services, the functions of MLEs and GUIs are the same.
3. Instance Representation (or just instance). It is generated by various optimization system components and exchanged among them. For example, an MLE parses a model and generates a problem instance. This problem instance is then sent to a solver to be solved. The instance component differs from other physical components in that it is a data piece rather than software.

4. Communication Agent/Interface. Communication agents are in charge of communication in a distributed system. No agents are needed in a local environment, in which case interfaces and objects are instantiated in memory and methods are invoked locally. Communication agents are used to send and receive instances. Instance representations and communication agents are least visible to system users, although they constitute the backbones of an optimization system.
5. Server/Registry. A server or registry is the heart of a distributed system. Optimization Services introduces the concept of an optimization registry (Figure 2-2). A registry as a lightweight server in that registry contains information *about* the software, but not the software itself. An agent first contacts the registry for location information about solvers. Upon response from the registry, the agent takes a second step to contact the solver in a peer-to-peer mode. In both steps, data representation and communication follow the Optimization Services Protocol. Such an arrangement alleviates the burden of any traditional optimization server. Another direct result of the decentralization is that solver providers will correspondingly assume a more independent role to compete for customers' business. We envision decentralization as the future in distributed optimization for it provides an encouraging environment for the development of optimization systems and components.

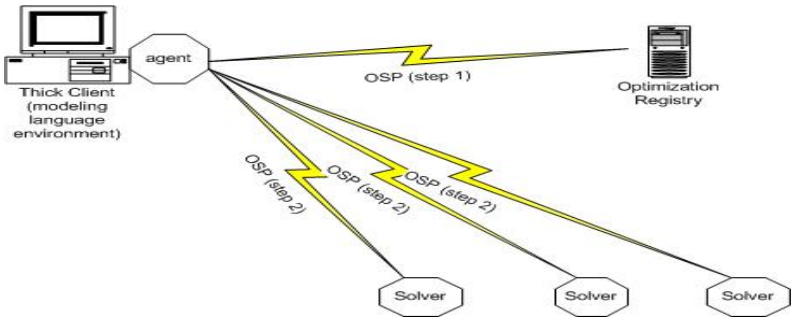

Figure 2-2: The optimization registry architecture.

6. Analyzer. Analyzers become a highly integrated and critical part of the OS framework, mainly due to the introduction of the optimization registry and the corresponding discovery mechanism in a decentralized optimization system. The output of an analyzer can be used by a solver query engine to locate the appropriate solvers for the model analyzed by the analyzer. Without analyzers, an optimization system can potentially involve much human interaction. So analyzers play a key role in automation.
7. Solver. Being the real "contents" of an optimization system, solvers make the whole system meaningful and are what users really need. Any solver on an Optimization Services system should take the Optimization Services instance Language (OSiL) as its input and generate the Optimization Services result Language (OSrL) as its output. A solver, however, does not usually carry out computation directly on the instance representation. Rather an instance reader parses the input into the internal objects or data structures required by the solver's algorithm. Optimization Services provides libraries for reading the standard OSiL input (OSiLReader) and writing the OSrL output (OSrLWriter), as shown in Figure 2-3.

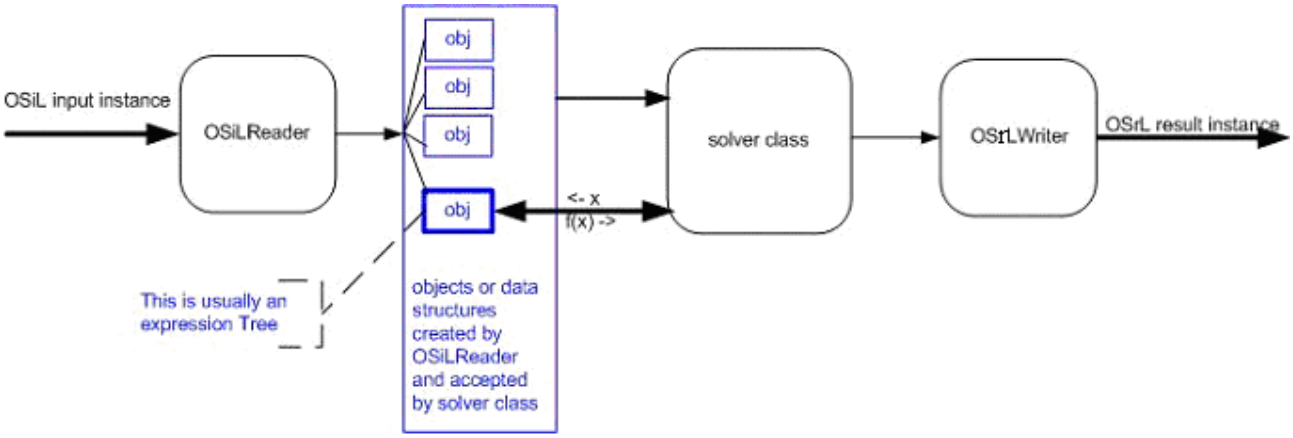

Figure 2-3: A generic input and output process of an Optimization Services compatible solver.

8. Simulation. A simulation is a black box function evaluator. The simulation engine may or may not reside with the solver. If the simulation is a simple function that stays locally with the solver, it is usually called a function evaluator, a function pointer, an evaluation routine, or an expression tree. In an optimization system, simulations are usually invoked by a solver. Most of the optimization solver algorithms involve some iterative schemes and each iteration may potentially involve an invocation of the simulation.

There can be many other components in the optimization system such as problem repositories and solver benchmarkers. However the above eight components are the key ones. They are the main targets to be "regulated" by the Optimization Services framework. When all these components are built according to the Optimization Services (OS) framework, we call them "OS-compatible," and we call the optimization system an Optimization Services system.

2.2 Optimization Services Process

The process of the optimization system in Figure 2-1 is self-explanatory. Typically the process starts from a modeler who has a model (1) to be solved. He constructs the model in an MLE (2). The MLE in turn compiles the model and generates an instance representation (3). The MLE then delegates a communication agent (4) to send the instance to a solver (7). In a local environment, where there is no agent, the link between 4 and 7 is an interface through which the MLE instantiates the solver in memory. In a distributed environment, the MLE may first discover a registered solver through registry (5), therefore the respective discovery link between 4 and 5 and registration link between 5 and 7. The communication with the analyzer (6) is similar to that with a solver (7). The link between communication agent (4) and simulation (8) means that the agent may call the simulation to get a function value, although in an optimization scenario, it is usually the solver (7) that calls the simulation (8) iteratively.

The triangle between communication agent (4), registry (5) and solver (7) is called a Service-oriented Architecture (SOA). The design philosophy of SOA serves as the basis of our Optimization Services

framework. A "service" is intended to serve customers. For our optimization system, there are mainly three categories of "human" customers:

- **Application developers** create and build system components such as modeling language environments and solvers as part of a larger optimization system. The components together take care of such generic functions as managing data, solving optimization problems, and presenting solutions nicely.
- **Modelers** work in a modeling language environment or GUI to build optimization models and get acceptable solutions.
- **Users** run application packages that perform optimization at some stage through the optimization system. Users are usually the ultimate customers of any optimization system.

Modelers and application developers may see optimization in different ways. For modelers, a mathematical program is an abstract representation to be analyzed and understood; for application developers, a mathematical program is a concrete instance to be represented, communicated and solved. Modelers benefit most immediately from innovations that help people to choose and experiment with optimization software. Some application developers are also modelers, while others deal mainly with the inputs and outputs of optimization models set up by modelers. Users may not even realize that they are running optimization system components such as solvers, although they are often aware of optimization goals, such as minimizing costs or maximizing profits.

The decentralized architecture of Optimization Services makes it similar to the Internet (Figure 2-4).

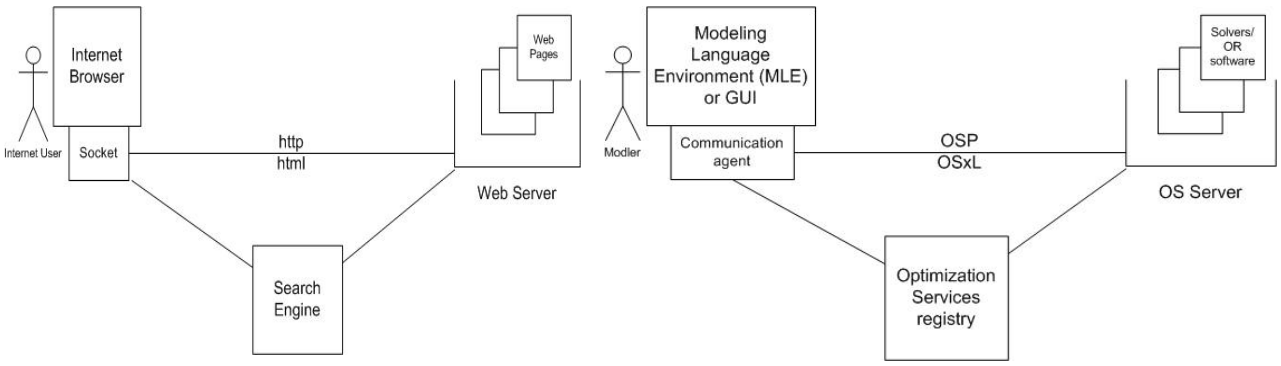

Figure 2-4: Similarity between the Internet and the Optimization Services architectures.

In order to "surf the Internet" (Figure 2-4, left), a user uses an Internet browser to view the Web pages, which usually contain interactive links and forms. Clicking the links and filling in forms are what we call the user inputs. In the scenario of Optimization Services (Figure 2-4, right), the user is a modeler and his inputs are a model and the model's data. Instead of the browser, the modeler constructs the model in a Modeling Language Environment or in a Graphical User Interface (GUI) environment and instead of sending the model inputs to a web server, the MLE or GUI sends the inputs to an OS server. The OS server hosts solvers rather than Web pages. Although the Internet existed long before it became popular, the entertaining Web pages

were what made the Internet successful. The same can be said about Optimization Services. Without the actual "contents" provided by the solvers, OS is just an empty skeleton that can never be widely used no matter how well the skeleton is designed.

To further apply the analogy, it is never the browser that contacts a web server. Rather the browser opens a socket, and through the socket, the browser sends the request and waits for the response. These all happen without the user's knowledge. The exact equivalent of the socket in Optimization Services is the communication agent. The MLE or GUI delegates the agent to send an optimization instance to the remote OS server that hosts the solver. Like the socket, the agent understands all the communication protocols in order to establish the connection. But instead of using the HTTP protocol and sending/receiving HTML instances, the agent uses the OSP communication protocol and sends/receives OS instances.

Nowadays people heavily rely on search engines to find Web pages. The Optimization Services registry serves the function of a search engine. But unlike the Internet search engines, there has to be a unique registry in the whole Optimization Services system to ensure Quality of Service (QoS). Communication agents always know where the registry is, as there is only one. This registry has complete information of available services, as this is the only place that the services can register. The OS registry will not be overburdened as no software is connected through it. When a certain query is sent to the OS registry, usually from an MLE or GUI, the OS registry returns the locations of the found software and the MLE or GUI makes a peer-to-peer contact with the software at the provided location. This discovery process is similar to the search engine process, with the exception that everything in the OS system happens automatically between the software components, without user interaction.

On the opposite side of the discovery process is the registration process. In the case of the Internet, it is usually the search engine "crawlers" that automatically collect the contents of all the Web pages. In the Optimization Services case, it is the OR software developer's responsibility to send the required information to, and get approved by, the OS registry, possibly through a mixture of automatic and manual procedures. This is primarily due to two reasons. One is that the quantity of OR software packages is not nearly large enough to be crawled efficiently. A second, and more important reason, is that the requirement of QoS on the OS registry is much stricter in order to ensure smooth functioning between OS components. The mechanism of "wantonly" crawling and storing "unwarranted" things found on the hyperlink paths degrades the Optimization Services.

In Figure 2-5, we show that most of the components in the Optimization Services system have a corresponding similar part in the Internet architecture. The similarity is not the initial intention of the OS project; rather it is the result that both are good designs based on a decentralized architecture.

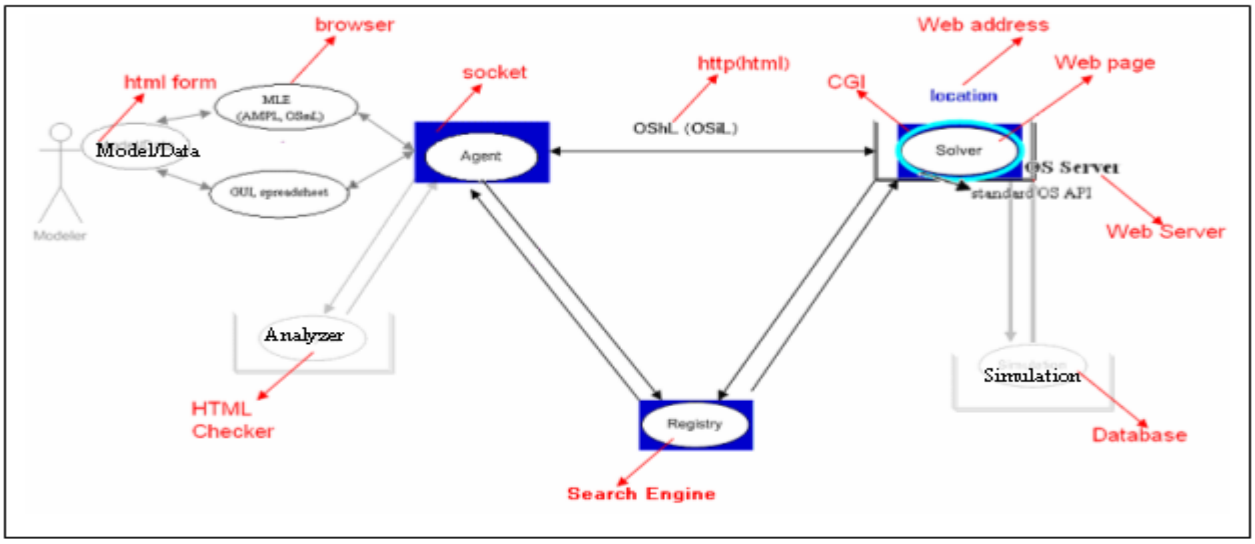

Figure 2-5: A close analogy between Optimization Services and Internet.

2.3 Standardization, OSP and OSxL

More involved than the HTTP/HTML protocol, Optimization Services Protocol contains different sub-protocols as there are more heterogeneous components than just the Web browser and Web server. The Optimization Services framework is mainly concerned with standardization in three areas: 1) Optimization (instance) representation (§2.4); 2) Optimization communication that includes accessing, interfacing and orchestration (§2.5); 3) Optimization service registration and discovery (§2.6).

For the sake of uniformity, we specify Optimization Services Protocols in all these three areas by standard 4-letter acronyms of the form OSxL, standing for Optimization Services x Language, where "x" is another defined letter. Figure 2-6 shows a tree view of all the current Optimization Services x Languages.

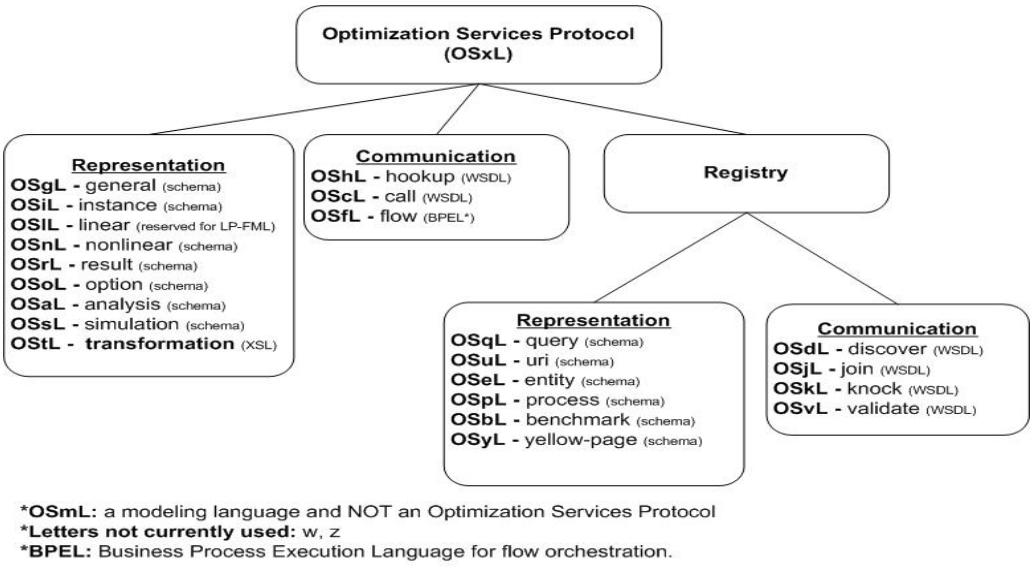

Figure 2-6: A tree view of Optimization Services x Languages (OSxL).

We briefly explain the OSxL languages in each of the three areas in the following sections.

2.4 Optimization Services Representation

In section 2.1, we discussed the differences between a model and an instance. The Optimization Services framework is *not* intended to standardize high level models. The framework only concerns itself with the low level representation communicated between machine and software components. All the instance representations are specified in the XML Schema language.

The most important instance is the representation of an optimization problem. The format of this instance is specified by the Optimization Services instance Language (OSiL). An OSiL instance is usually transmitted from a modeling language environment to a solver.

The OSiL language is the first effort toward a systematic and universal representation of major optimization types. Currently OSiL supports all major optimization types including mixed integer programming, quadratic programming, general nonlinear programming, complementarity problems, optimization with distributed and real-time data, optimization with user defined functions, simulation optimization, cone programming, constraint programming, combinatorial problems, multi-objective optimization, semidefinite programming, linear/nonlinear stochastic programming and networking programming. OSiL is targeted to replace many of the existing (some outdated) standards such as MPS for linear and quadratic programs, and SMPS for linear stochastic programs. One thing to notice is that before OSiL, there is not a standard instance format for general nonlinear programs. Solvers usually take the instance generated from a modeling language, and use the library provided by the MLE to parse the instance.

Besides OSiL, there are other kinds of instances. The Optimization Services result Language (OSrL) specifies the result format of the solver output. It is usually transmitted back from a solver to an MLE. Optimization Services analysis Language (OSaL) specifies the analysis format of the analyzer output. It is usually transmitted from an analyzer to an MLE and helps in discovering solvers in an Optimization Services registry. Optimization Services option Language (OSoL) specifies the option format of solver (or analyzer) algorithm directives. It is usually transmitted along with an OSiL instance. Optimization Services simulation Language (OSsL) specifies the input and output format of a simulation service. It is usually transmitted between a solver and a simulation engine. It facilitates optimization via simulation where simulations are located in places other than the solver.

Many of the generic and common data structures are specified in the Optimization Services general Language (OSgL). OSgL is then imported by other representation schemas. All the nonlinear functions, operators and operands are specified in the Optimization Services nonlinear Language (OSnL). OSnL is then imported by the OSiL schema for nonlinear optimization extension. The OSnL schema is very comprehensive; over 200 elements are supported. They fall broadly into the following 8 categories:

1. Arithmetic operators such as `plus`, `minus`, `power`, and `sum`.
2. Elementary functions such as `abs`, `square`, `log`, and `exp`.
3. Trigonometric functions such as `sin`, `sec`, `csc`, `cosh`, `arctan`, and `arcsech`.
4. Statistical and probability functions such as `min`, `skewness`, `percentile`, `covariance` and common density, cumulative, and inverse distribution functions.
5. Terminals and constants such as `number`, `string`, `PI`, `E`, `TRUE`, `INF`, and `NAN`.
6. Optimization related elements such as `var`, `constraint` and `objective`.
7. Logic and relational operators such as `lt`, `geq`, `and`, `xor`, `if`, `forall`, `exists`, and `atLeast`.
8. Special elements such as `quadratic`, `userF`, `sim`, `xPath`, and `complements`.

2.5 Optimization Services Communication

The Optimization Services framework standardizes all the communications between *any* two Optimization Services components on an OS *distributed* system. The framework itself does *not* standardize *local* interfacing. Derived researches from Optimization Services (§4.6) are intended to do this job. The Optimization Services framework is complementary to the standardization of local interfaces. The connection between Optimization Services and local interfacing is illustrated in Figure 2-7.

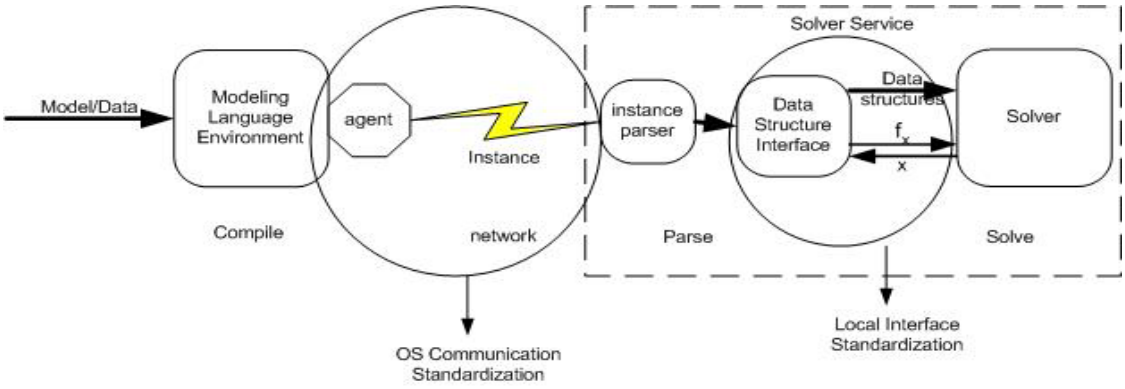

Figure 2-7: Relationship between OS Communication and local interface standardization.

Standards for optimization instance communication over *distributed* systems are new. But the standardization is technologically timely. Distributed technologies such as Web services are growing rapidly in importance in today's computing environment and are already widely accepted as industrial standards. It is our vision that by combining Operations Research and modern distributed technologies, Optimization Services will make a wider audience able to easily access and benefit from the increasing number of OR software packages. Through standardization of communication, the OS framework provides an open infrastructure for all optimization system components to communicate with each other. The goal is that all the algorithmic codes will be implemented as services under this framework and customers will use these computational services like utility services. Special knowledge of optimization algorithms, problem types,

and solver options required of users should be minimized. Everything that involves finding the right solver, invoking the software, providing the computing resources and presenting the solution is automatically taken care of by Optimization Services.

Invocations of all Optimization Services are specified by Web Services Definition Language (WSDL [15]) and all the interfaces and transport parts (i.e. except for the location information) in the WSDL documents are standardized. So WSDL documents are not necessarily needed to *dynamically* generate the communication APIs (stubs and skeletons) as we know them ahead of time already, although they can be used for illustrations or as references to construct Optimization Services *beforehand*.

The most common communication is the invocation of solvers. This is specified by the Optimization Services hookup Language (OShL). OShL also applies to hooking up to analyzers, as solvers often analyze an optimization problem and analyzers may potentially solve the problem. The invocation of simulation services is essentially calling a function and is specified by the Optimization Services call Language (OScL).

Communication is not just about invocations. As we build all the Optimization Services components into a distributed system, the *sequence* of invocations is an issue. For example, if a solver service is known to a client, the client can directly contact the service. Of course the client can still contact a registry and get the location information and then call the solver. But if the client does not know the type of the optimization problem, he may first invoke an analyzer service, and then use the analysis result to query the right solver from the registry. Even more complex, before invoking the analyzer service, the client may need to find the analyzer's location in the registry. There can be many combinations of sequences. Optimization Services flow Language (OSfL), an XML document in BPEL (Business Process Execution Language [7]), predefines certain standard flows.

2.6 Optimization Services Registry

To locate services in a decentralized serviced-oriented distributed system, software agents coordinate with each other and with registries. Some registries are general ones that keep information of all kinds of Web services, such as Universal Description, Discovery and Integration (UDDI, [10]). Others are specialized ones like the OS registry that only serves registration and discovery of Optimization Services. The OS registry knows all the registered services (solvers, analyzers, simulations) on the OS network by keeping their metadata information. The OS registry can be viewed as a light-weight server as no registered services are actually executed by this registry; instead clients directly contact the services in a peer-to-peer mode.

In terms of standardization, OS-registry related protocols do not face as much imminent pressure of universal acceptance as the non-registry related protocols. There is only one public registry on the entire system and there are much fewer registry developers compared with the other OS developers.

There are mainly two categories of registry-related OSP protocols; one deals with representation and the other deals with communication. To ensure that the OS registry only sends addresses of the services (especially solvers) that are of reasonably high quality, regulations are imposed when an OS-compatible service is to be registered in the OS registry. The following three OSP protocols are designed to make sure a solver is and continues to be well-described, live, reliable, and robust. Information about registered services in the OS registry includes three main categories:

1. Entity information that is reported by service developers at registration, e.g. service and owner information, solver or simulation types and service locations. We call this category of information "entity" information to emphasize the information is relatively static. This is addressed by the Optimization Services entity Language (OSeL).
2. Real-time process information that is either reported by the registered service ("push") or detected by the OS registry ("pull"), e.g. how many optimization jobs are at the service server. We call the information "process" information to emphasize the information is dynamic. This is addressed by the Optimization Services process Language (OSpL).
3. Benchmark information that is gathered separately by auxiliary benchmarker tools designated by the OS registry, e.g. general solver ratings and performance profiles. This is addressed by the Optimization Services benchmark Language (OSbL).

All the three types of information are kept in an XML database of the OS registry. As the OS registry is an open registry, to facilitate communication (especially discovery) with the registry, the structure and contents of the OS database are made public just like a yellow pages directory. The structure and contents in the OS database are addressed by Optimization Services yellow-page Language (OSyL). To query the database, clients use the Optimization Services query Language (OSqL). In the OS registry implementation, an OSqL query is then converted to an XQuery [16] that is executed against the XML database in the registry. The communication of sending the OSqL query to the OS registry is specified in the Optimization Services discover Language (OSdL). In turn the clients get the location information from the registry that is listed as a sequence of URIs (or URLs); the syntax is specified in the Optimization Services uri Language (OSuL). The discovery mechanism is similar to sending an SQL query to a relational database.

On the other side of the *discover* process is the *register* process. Service providers join the registry with OSeL information. Optimization Services join Language (OSjL) specifies how this is done. During runtime, the Optimization Services registry periodically "knocks" on the registered services to make sure they are live and running and to get the OSpL information. Optimization Services knock Language (OSkL) specifies how this is done. Service providers can also publish the entity, process, and benchmark information on their own Web site. To facilitate standardization, the standard style sheet OStL (Optimization Services transformation Language) is provided so individual Web publications have the same look-and-feel.

3. OPTIMIZATION SERVICES MODELING LANGUAGE (OSmL)

OSmL is among the many derived researches (§4) from the Optimization Services project. OSmL is a computer modeling language for mathematical optimization. This language allows mathematical model developers to formulate complex and large-scale optimization problems in a concise and efficient way. OSmL is based upon the W3C XQuery standard and is designed to convert raw data in XML format into problem instances that conform to the Optimization Services instance Language (OSiL) standard. An optimization instance represented in OSiL can be solved with any standard solver that is Optimization Services compatible. Thus, OSmL is particularly well suited for optimization over distributed systems.

Notice that OSmL itself is *not* an XML dialect, but rather a customized implementation of XQuery. XQuery provides a concise query language and is designed to quickly and efficiently extract chunks of data – much like SQL for relational databases. Unlike other OSxL languages, OSmL is *not* a low-level instance language. OSmL is a high-level user friendly *modeling* language. An advantage of the OSmL approach is that people can easily share and reuse OSmL models regardless of computing platform. All of the required software is open source and available on all major platforms.

3.1 Motivation

XML [12] is popular and powerful and is rapidly becoming an accepted format for transferring and storing data. Currently almost all databases and spreadsheets are xml-enabled. This means that even if the stored data are not in native XML form, they can at least be exported in XML formats. The XML output can then be retrieved with XQuery. One might argue that mathematical modeling is also about data. Indeed, a mathematical programming modeling language, and associated solver tools, will not be used unless they are closely integrated with corporate data. The creation of the OSmL modeling language is motivated by two factors. First is the ubiquity of XML and the existence of tools to easily transform the non-XML data into an XML format. Second is the availability of many powerful open source platform independent tools for taking XML data stored in one format and transforming it into another XML format.

3.2 Four Paradigms of Combining XML with Optimization

It is common practice to store data in a relational database system. Two aspects of commercial relational database systems are 1) the data are stored in multiple tables or relations, and 2) the files containing the data are typically binary files. XML data is 1) stored using a tree structure, and 2) stored as a text file containing both tags and the data.

There are four paradigms to incorporating XML into mathematical modeling of optimization problems:

1. Use XML to represent an optimization instance; this is what Optimization Services is all about.
2. Develop an XML modeling language dialect; this is what Optimization Services avoids.

3. Enhance modeling languages with XML features such as XPath [17]; this is a recommendation from Optimization Services to enable existing modeling languages to access XML data.
4. Use XML technologies to transform XML data into a problem instance; this is the approach taken by OSmL, which is a derived research of Optimization Services.

The OSmL approach is to take as input the XML files that contain the problem instance *data* and then transform the input data files into a standard OSiL instance output. OSmL leverages on XQuery for the data transformation. XQuery provides powerful algebraic modeling features, e.g. sets, loops, if-then, union, intersection, and library modules. These are accepted W3C standards and are what makes the OSmL modeling language symbolic, general, concise and understandable [3].

The OSmL model compilation and OSiL instance generation process are illustrated in Figure 3-1. We have been emphasizing the fact that OSmL is XQuery based, but OSmL is a bit more than an XQuery language. OSmL has extra pre-built constructs tailored for optimization problems. For example, the relational operators "<", "<=", ">" and ">=" are represented in XQuery as "`>`", "`>=`", "`<`" and "`<=`" to avoid conflicts with the XML tags (< >). As the relational operators appear very often in optimization, OSmL allows users to use "<", "<=", ">" and ">=" directly and it has a "preprocessor" to detect these operators and convert them to the XQuery language specification. Also OSmL adds some macros such as the "`SUM`" function, as again these macros provide extra convenience for mathematical modeling. The OSmL preprocessor expands these macros to XQuery equivalents. A modeler can still directly use the standard XQuery representations to construct an optimization model without the OSmL macros. In this situation, OSmL is a pure XQuery language. A pure XQuery (original or preprocessed) is sent to an XQuery processor and compiled into an intermediate XML instance. The immediate instance is converted into the OSiL instance and sent out to an Optimization Service using a standard OS communication agent.

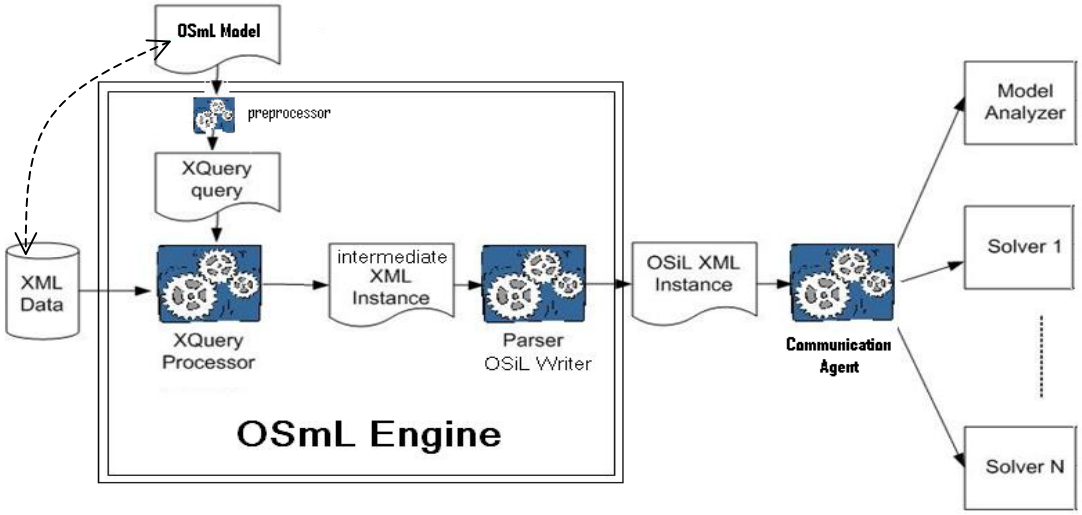

Figure 3-1: The OSmL process.

4. DERIVED RESEARCH FROM OPTIMIZATION SERVICES

4.1 The Optimization Services Project

Optimization Services is a young research area that has potential for many benefits to operations research and the optimization community. Motivated by a vision of the next generation of optimization software and standards, Optimization Services deals with a wide variety of issues that have accumulated over the past few decades in computing and optimization. This work addresses design as well as implementation issues by providing a general and unified framework for such tasks as standardizing problem representation, automating problem analysis and solver choice, working with new Web service standards, scheduling computational resources, benchmarking solvers, and verification of results – all in the context of the special requirements of large-scale computational optimization. The criteria required of Optimization Services must therefore be very high. Improving the quality of OS related standards, tools and systems should be a constant effort. Adapting to the new needs of researchers (scalability) and best serving the ultimate users (ease of use) should always be the goal of OS. Collaborations have been established in many of the following areas.

4.2 Standardization

Optimization Services involves a large set of standard protocols that need to be adopted quickly and universally. The standardization process starts from working group notes, and goes through stages such as working drafts, candidate recommendations, and finally becomes recommended as standards. Such a process not only requires further research efforts such as new optimization problem extensions but also entails more organizational efforts that require formal establishment of collaborations under the OS framework.

4.3 Problem Repository Building

With the standardization of various problem representations naturally comes the task of building repositories of optimization problem instances using the OS standards. Problem repositories no longer need to be categorized by the format the problems are using. Rather they are only classified by the different optimization types supported in the OS standards.

4.4 Library Building

The OS library and the OS server are provided to facilitate the adoption and use of the OS standards. Besides the original OS designers, other researchers are free to develop their own OS-compatible libraries, such as readers and writers of standard instances, and communication agents to transmit these instances.

4.5 Derived Research in Distributed and decentralized Systems

A distributed system leaves open many questions in coordination, job scheduling and congestion control. One distinct issue for example is how optimization “jobs” should best be assigned to run on

available registered services after the optimization types are determined. The usual centralized scheme of an optimization server maintains one queue for each solver/format combination, along with a list of the workstations on which each solver can run. In a decentralized environment, we may still want to maintain this scheduling control, while at the same time making the scheduling decisions more distributed, i.e. transferring some controls to the solver service sides. Further study is needed to better understand how categorization of optimization problem instances together with statistics from previous runs can be used to improve scheduling decisions. As just one example, an intelligent scheduler should not assign two large jobs to a single-processor machine, since they will only become bogged down contending for resources; but a machine assigned one large job could also take care of a series of very small jobs without noticeable degradation to performance on either kind of job. Both the kind and size of optimization instances must be assessed to decide which should be considered "large" or "small" for purposes of this scheduling approach.

The central issue in a decentralized architecture is the design of a registration and discovery mechanism for acting on service requests. For example the optimization registry could assign requests based on some overall model of solver performance and resource availability. Requests can be scheduled after they are matched to some services, or scheduling could be made an integral part of the assignment process. Pricing could involve agent "rents" as well as charges determined by various measures of resource use. Besides keeping and maintaining information on optimization solvers and other services, one critical and more complex role of an optimization registry in a decentralized environment is a "more confident" determination of appropriate solvers. A relatively easy and straightforward scheme can rely on a database that matches solvers with problem types they can handle. Characteristics of a problem instance, determined from the analyzers, can be used to automatically generate a query on the database that will return a list of appropriate solver services. But how should solver recommendations deal with problem types (e.g. bound-constrained optimization) that are subsets of other problem types (e.g. nonlinear optimization)? Or how can recommendations be extended to solver options? For these purposes, a straightforward database approach for a server or registry may not be adequate. Developers will consider more sophisticated ways of determining recommendations, such as through business rules systems. A more complicated and advanced scheme may consider extensions to generate lists ranked by degree of appropriateness.

4.6 Derived Research in Local Systems

The Optimization Services framework standardizes all the communications between *any* two Optimization Services components on an OS *distributed* system. The framework does *not* standardize *local* interfacing. Related projects such as COIN [1] and derived research from Optimization Services such as the Optimization Services instance Interface, Optimization Services option Interface, and Optimization Services result Interface are intended to do this job. The COIN project includes the OSI (Open Solver Interface)

library which is an API for linear programming solvers, and NLPAPI, a subroutine library with routines for building nonlinear programming problems. Another proposed nonlinear interface by Halldórsson, Thorsteinsson, and Kristjánsson is MOI (Modeler-Optimizer Interface [5]), which specifies the format for a callable library. This library is based on representing the nonlinear part of each constraint and the objective function in post-fix (reverse Polish) notation and then assigning integers to operators, characters to operands, integer indices to variables and finally defining the corresponding set of arrays. The MOI data structure then corresponds to the implementation of a stack machine. A similar interface is described in the LINDO API manual [6]. The OS framework is complementary to all of the standardization of local interfaces.

4.7 Derived Research in Optimization Servers

Optimization Services is motivated by the current issues faced by many optimization servers. More specifically Optimization Services is intended to provide the next-generation NEOS [2]. The effects of Optimization Services on NEOS are multifaceted:

- The NEOS server and its connected solvers will communicate using the Optimization Services framework, e.g. using standard representation for data communication.
- External optimization submissions can still be kept as flexible as possible and may become even more flexible. At least one more networking mechanism will be provided, i.e. the communication based on the Optimization Services Protocol (OSP). That means NEOS will add an interface so that it can be invoked exactly as what's specified by the Optimization Services hook-up Language (OShL). It will also accept OSiL as a standard input, and may gradually deprecate the other formats.
- The entire OS system can be viewed as a new decentralized NEOS. In effect the old NEOS will become another OS-compatible solver in the new system. The "NEOS solver" can then solve more types of optimization by delegating the job further to different solvers behind it. We therefore regard the old NEOS as a "meta-solver" registered on the new OS system.

4.8 Derived Research in Computational Software

With the advent of Optimization Services and its standard OSP protocol, related software developers may need to think about how to best adapt to the OS framework and be "OS-compatible." There have already been two immediate projects that are related to the Optimization Services framework. One is the Optimization Services modeling Language described in Section 3 and the other is the IMPACT solver development project that is under development by Professor Sanjay Mehrotra's group at the Industrial Engineering and Management Sciences department at Northwestern University. The two projects are the two sides of Optimization Services: client and service. Both are natively built *for* the Optimization Services framework and strictly follow the Optimization Services Protocol. There are existing modeling languages and solvers that are or will be adapted (by writing

wrapper classes) to the Optimization Services framework such as the AMPL modeling language, Lindo API [6], open COIN software [1] and solvers from the NEOS system [2].

4.9 Derived Research in Computational Algorithms

The design of effective and efficient computational algorithms that fit the Optimization Services design is important. Optimization Services immediately opens up the questions of how to best utilize the available services on the OS network. Following are some of the potential research areas in computational algorithms related to Optimization Services:

- Parallel computing where many services can simultaneously solve similar optimization problems.
- Optimization via simulation where simulation services are located remotely from OS solvers.
- Optimization job scheduling at the registry side and queuing at the service side.
- Analyzing optimization instances according to the needs of the OS registry.
- Modeling and compilation that generates OSiL instances quickly and accurately.
- Efficient OSxL instance parsing and preprocessing algorithms.
- Effective Optimization Services process orchestration.
- As the OS standards allow representations of various optimization types, algorithm development (e.g. stochastic programming) that has lagged due to lack of good representations can hopefully get a boost.

4.10 Commercialization and Derived Business Models

Optimization Services, though itself an open framework, does not prevent registered services and related business to be commercialized. Following are some of the related business models:

- Modeling language developers can leverage on using Optimization Services to provide more and better solver access to their customers and become more competitive.
- Solver developers concentrate on developing better algorithms to increase their competitiveness without worrying about representation, communication and interfacing that are taken care by the OS standards.
- Developers can commercialize their OS libraries, e.g. readers and writers of standard instances.
- Registry/server developers can provide auxiliary services such as storage services, business process related flow orchestration services, advertisement services and consulting services.
- Auxiliary services such as analyzers and benchmarkers may possibly charge fees to involved parties.
- Solver owners may adopt a “computing on demand” model by charging the user for using their services.
- A solver service owner may also adopt a “result on demand” model by reporting the objective results that his solver service has found but hiding the solutions that are only to be revealed when the user agrees to pay. For example in the Optimization Services result Language (OSrL) that the solver service returns, it may write out only the `<objectiveValue>` value and in the `<solverMessage>` element it may provide the payment instructions for obtaining the `<variableSolution>` values.

REFERENCES

[1] COmputational INfrastructure for Operations Research (COIN-OR), http://www.coin-or.org (2005).

[2] E.D. Dolan, R. Fourer, J.J. Moré and T.S. Munson, "Optimization on the NEOS Server." SIAM News 35, 6 (2002) 4, 8-9.

[3] R. Fourer, Modeling Languages Versus Matrix Generators for Linear Programming. ACM Transactions on Mathematical Software 9 (1983) 143-183.

[4] R. Fourer, Next Generation Servers for Optimization as an Internet Resource, http://users.iems.nwu.edu/~4er/NEOSprop.pdf (2003).

[5] B.V. Halldórsson, E.S. Thorsteinsson and B. Kristjánsson, A modeling Interface to Nonlinear Programming Solvers – An Instance: *x*MPS, the Extended MPS Format. Technical report, Department of Mathematical Sciences and Graduate School of Industrial Administration, Carnegie Mellon University (2000).

[6] Lindo Systems, Inc., API user's manual, Tech. Rep., Lindo Systems, Inc. http://www.lindo.com/lindoapi_pdf.zip (2002).

[7] OASIS, BPEL, http://www.oasis-open.org/committees/wsbpel/charter.php (2005).

[8] Optimization Services, www.optimizationservices.org (2005).

[9] Optimization Services, www.optimizationservices.net (2005).

[10] UDDI.org, Universal Description, Discovery, and Integration (UDDI), http://www.uddi.org (2003).

[11] W3C, World Wide Web Consortium, http://www.w3.org (2005).

[12] W3C, XML, http://www.w3.org/XML (2005).

[13] W3C, XML Schema, http://www.w3.org/XML/Schema.html (2004).

[14] W3C, Web Services, http://www.w3.org/2002/ws/ (2005).

[15] W3C, Web Services Description Language, http://www.w3.org/TR/wsdl (2005).

[16] W3C, XML Query Language (XQuery) http://www.w3.org/TR/xquery (2005).

[17] W3C, XML Path Language (XPath) http://www.w3.org/TR/xpath (1999).