OOF: Open Optimization Framework

Obi C. Ezechukwu
Department of Computing
Imperial College London
Exhibition Road
London SW7 2BZ
United Kingdom

mailto:oce@doc.ic.ac.uk

Istvan Maros
Department of Computing
Imperial College London
Exhibition Road
London SW7 2BZ
United Kingdom

mailto:im@doc.ic.ac.uk

Departmental Technical Report 2003/7

ISSN 1469-4174

April 2003

Abstract

There is currently a plethora of formats for representing optimization models and instances. The varying degrees of support for these formats, coupled with their well-documented deficiencies complicate the task of developing and integrating optimization models and software. This has led to new initiatives to develop new forms of model representation, which address these deficiencies and exploit advances in software technology, particularly the Internet. This paper describes a framework, which not only comprehensively addresses the issue of model and instance representation but also provides in-built support for distributed optimization, particularly in the area of service delivery over the Internet.

1 Introduction

The popularity of mathematical programming has grown a great deal, since its early years. This growth has been followed and driven by great advances in the field including new solution techniques, cheap computing platforms, and a vast array of software tools. Among these software tools are solution algorithm implementations i.e. solvers including those embedded in popular spreadsheet applications. These have helped to greatly simplify the implementation and solution of optimization models, thereby fuelling the acceptance of mathematical programming.

However, technology, software and standards related to model representation have advanced at a much slower rate compared to technology for solving and analyzing models once they are created. If at all, the highly fractured nature of model representation and modelling in general has frustrated researchers and developers in the field of mathematical programming. This is due to the very low level of portability, and re-use of optimization models and related software, and also the high cost of model implementation. Moreover most real world model implementations are less likely to run standalone and are more likely to be embedded in, or linked to an application, most commonly a DSS *(decision support system)*—a computer based technology for representation and computation of data and models in order to gain insight into decision problems. Therefore the lack of flexibility of model representations has also hindered the widespread commercial adoption of mathematical programming.

Furthermore existing technologies have failed to take advantage of recent advances in computing particularly in the areas of the programming languages, the Internet, web services [34] and more generally distributed computing [2].

In this paper we propose a formal and open-source framework which seeks to address the problems associated with model representation. This framework proposes structured and portable formats for representing optimization and constraint programming models, instance data, and result information. It also provides a recommendation which abstracts the process of invoking operations research software—a collective term which refers to optimization and constraint programming software. The aim of this framework is not to define yet another format or standard for representing models, but rather to propose a scheme, which abstracts away from the concept of representational formats and allows the portability of models from one format to the other. The framework also includes a recommendation which will enable application developers and vendors to take advantage of advances in programming language design and distributed computing particularly in the area of Internet computing.

The two key driving factors behind the initiatives to create a new class of model representation formats are: the problems associated with existing representational formats; and a desire to exploit advances in computing technology, particularly in the area of the Internet. In order to fully explain these factors, a chapter is dedicated to model representation, and another to the opportunities which the Internet provides to operations research. In addition, the paper also introduces the core components of the framework, namely; *AML (Algebraic Mark-up Language) [12]*—a mark-up language which is used to describe optimization and constraint programming models, and instance data; *ORML (Optimization Reporting Mark-up Language) [13]*, which describes a syntax for representing result and analysis data for optimization and constraint programming model instances; *WSOP (Web Services Optimization Protocol) [15]*—a recommendation for distributing and accessing operations research functionality via basic Internet protocols; and *OSCP (Optimization Service Connectivity Protocol) [14]*—a Java recommendation which specifies contracts for dynamically locating and invoking operations research software at runtime.

2 Model Representation

In this section we explore the past, present and future of model representation. We provide a historical perspective on model representation which culminates with the current formats and standards for representing optimization models, including the limitations of these formats. Finally we look at ongoing initiatives aimed at creating new formats for representing instances of optimization models.

2.1 A Historical Perspective

Initially, mathematical programmers needed a way to express their models, and they did so with general purpose programming languages like FORTRAN or what were then called *matrix generators.* These matrix generators were in essence a library of sub-routines which were used to generate and solve instances of optimization problems. The library approach is still used today and notable libraries include IBM's OSL [25] and OR-Objects [39]. The latter of the two is a Java based library which contains implementations of a set of solution algorithms, and provides classes for representing model instances.

Some of the early *matrix generators* evolved further into primitive '*modelling languages' or* what we now refer to as input formats. At the time these represented the first step towards advanced modelling languages. These '*modelling languages'* were in reality formats utilised by these systems for model representation. These systems typically consisted of an implementation of a solution algorithm that relied on a specific format for problem input. An example of such a system is *IBM's MPS (Mathematical Programming System)* which eventually gave rise to the MPS [33] format for linear and integer programming. As the benefits of these input formats filtered into the OR community, they were adopted by more and more solution algorithm implementations—*solvers*. A number of these survived to the present day as proprietary or industry standards e.g. the MPS format for linear programming. This is due to the fact that a number of models particularly in the academic arena still exist in this format, also a number of applications still utilise MPS as the de-facto standard for representing model instances. In addition to the early formats, a whole new set of input formats have been defined to cater for various classes of problems, or to take advantage of special characteristics of specific problem instances. These include xMPS [52] which is a direct extension of the MPS format used to represent non-linear models, *SIF (Standard Input Format)* [42] also used for representing non-linear models. Support for these formats vary from one solver to another, and with the exception of MPS, majority of them are proprietary and non-standard.

Building on early work on input formats and solver technology, a new breed of *algebraic modelling languages/systems* emerged which abstracted away from the low level input formats. These simplified the task of model implementation, by enabling the expression of mathematical programs using symbolic notation. The symbolic representations are transformed into a low level format by a compiler, so that they can be read by a *solver,* which in turn produces solution text if and when a solution is found. This solution text is reformatted and displayed to the user. Commercial systems such as GAMS [45], AMPL [16] and MPL [32] usually consist of a compiler/interpreter component which is responsible for converting the algebraic model instance into a low level input format, and a stable of solvers which accept these compiled models and produce a solution if and when one is obtained. In addition to the core task of solving models, these systems also provide additional features such as database plug-ins for data management.

At present, programmers and modellers are left with two core approaches for implementing and representing optimization models. On one hand it is possible to use a high level algebraic

language such as GAMS or AMPL, and on the other it is possible to use a high level programming language such as C++ or C. With the latter approach, custom solution algorithms have to be coded, or special code has to be written to interface the model with an underlying solver or even an algebraic modelling environment.

In the next section we explore the problems associated with the various forms of model representation.

2.2 Limitations

It may be possible to argue that model representation in itself does not constitute a crucial aspect of the mathematical programming process, as a solution still has to be obtained from the representation. Also we may want to perform additional tasks before and after solving the model e.g. pre and post optimality analysis. However the importance of model representation is cemented by the fact that little can be done in terms of computation without a proper, accurate and computable representation of the model and its associated instances. With the exception of the academic domain, mathematical programs hardly exist in a standalone fashion. In a majority of cases they are embedded in decision support systems, or some other dependent application. After all, the ultimate aim of implementing/developing optimization based systems is to solve real life problems. Therefore, to fully appreciate the limitations of current forms of model representation we have to look at difficulties involved both in the implementation of mathematical programming models and decision support systems. The following paragraphs explore the various alternatives available to today's practitioner and highlight the problems associated with each.

2.2.1 Low Level Input Formats

Real world applications of mathematical programming involve solving problems on a large scale. Representing these problems using a low level format such as the MPS format is both time consuming and error prone. Moreover these formats do not distinguish between models and model instances. A *model instance* is a model where values have been assigned to the data structures or input parameters. Distinguishing between a model and an instance of a model increases the re-usability of the model, as it is possible to solve it with different sets of data, and at different times if necessary—e.g. when performing sensitivity analysis. Whereas a simple low level text format such as MPS may be suitable across multiple platforms, it is not the most re-usable. Moreover, there is a vast array of matrix level formats for representing optimization models, and support for these formats varies from one commercial solver to another. With the exception of the MPS format for representing LP and integer models, majority of formats are proprietary or non-standard, and as such reduce the scope of model implementations. Figure 2.1 illustrates this approach to model representation and solver interfacing.

2.2.2 Callable Libraries

It is possible to utilise callable libraries to communicate directly with solvers, an example of such a library is the CPLEX Callable Library [28]. This is illustrated in figure 2.2. These libraries in addition to solver interface routines also provide data structures for representing the model and its associated parameters. Whereas this relieves the burden of having to write code to generate the model matrix, it restricts the portability and flexibility of the implementation. This is because it effectively ties the implementation to a single solver or solution algorithm. If the solution algorithm is from a third party, it also introduces the prospect of vendor lock-in. With such an implementation it may be impossible to take advantage of advances in solution techniques, software and even hardware technology, or lower licensing costs. Arguably it is possible to implement some level of abstraction or driver for the callable libraries in use, so that they can be loaded at runtime depending on the

properties of the problem being solved. With such a scheme, if there are N potential solvers,

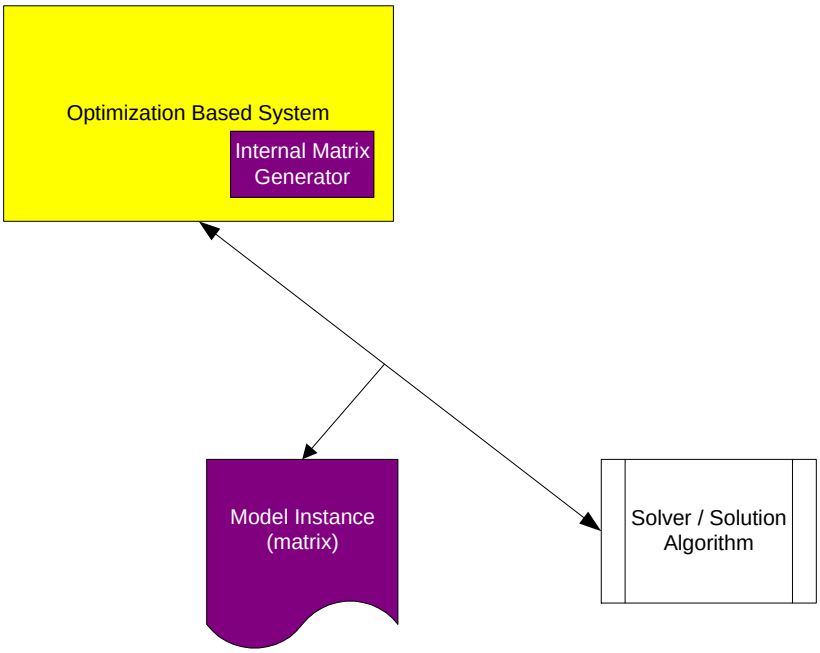

Figure 2.1: Implementation with an inbuilt matrix generator

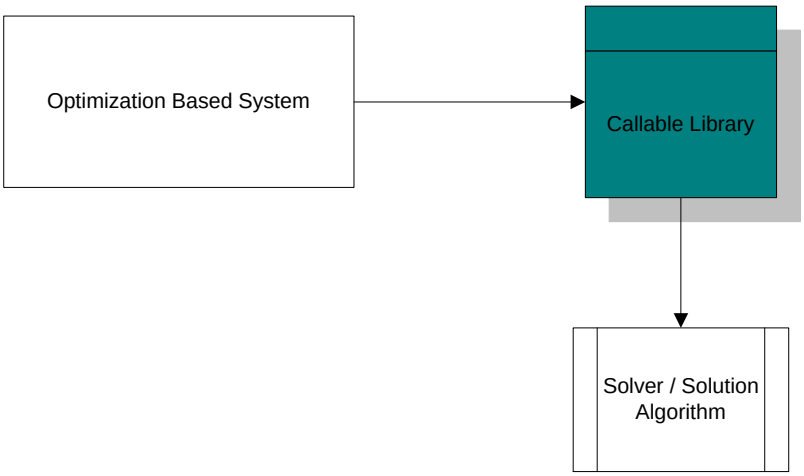

Figure 2.2: Interface based on callable library

then the work to integrate these solvers will have to be repeated N times. An exercise that is not only tedious and error prone but also potentially expensive. In most cases the effort involved in doing this far outweighs the benefits of switching solvers or solution algorithms, and as such is considered prohibitive. In either case, this shouldn't be done per implementation as it leads to the duplication of effort. In a number of other areas of computing, for example database programming, the realisation of the need to eliminate this duplication of work has resulted in a number of interface standards e.g. ODBC [37], JDBC [26], CORBA [8] etc. which abstract away from low level implementation interfaces.

2.2.3 Algebraic Modelling Languages

Although mathematical models may be implemented at a high level of abstraction using an algebraic modelling language e.g. GAMS, AMPL, these are not the most portable or re-usable

of formats. This is because such implementations are restricted within the particular modelling system which provides the algebraic language. This means that it may not be possible to share the model with another party, as the receiving party needs to have access to a similar environment in order to be able to execute the model. The model will have to be re-implemented each time it is ported to a new modelling system. Moreover as already mentioned, majority of model implementations exist as part of a wider system, hence using an algebraic language in such an implementation would more often than not require a *hybrid approach* as described below.

2.2.4 Hybrid Approach

It is possible to use an algebraic modelling system as the backend for a decision support system as opposed to using a single solver and or callable library. A scenario which would warrant such a scheme is where the system needs access to more than one solver. Implementing interfaces for multiple solvers can be a time consuming and expensive exercise, and in order to avoid this, developers can choose to interface to a modelling system instead thereby delegating the task or details of solver interfacing to it. However this is assuming that it is possible to find a single modelling environment with enough functionality to meet the requirements of the implementation. For non-trivial models it may be impossible to find a single modelling language or software with sufficient features to support an effective implementation. This is because most modelling software and solvers address just a few among the many classes of models that arise. This approach is highly inflexible at best and unworkable at worst, because it is more often than not necessary for models of different kinds to be integrated in order to address issues of importance.

It also introduces problems related to flexibility, scalability and vendor lock-in. Majority of organisations prefer not to limit their implementations to a specific vendor so that they can switch platforms as and when they wish to. This not only allows them to take advantage of advances in software engineering, but also gives them the opportunity to take advantage of cheaper licensing costs if the opportunity arises. The hybrid approach is illustrated by figure 2.3.

2.2.5 Custom Implementation

An alternative approach to model representation and solver integration is to implement custom model data structures and custom solution algorithms. Needless to say that this is the most time consuming of all the approaches, and raises serious questions about the re-usability, and flexibility of implementations based on it. This scheme is most likely to be found in high end research or commercial applications, and more often than not is the result of implementing a custom solution algorithm or solver which takes into account special properties of the problem being solved.

2.2.6 Common Limitations

In addition to the problems associated with integrating to a solver or underlying modelling or optimization system, existing model representation techniques have failed to keep up with advances in computing particularly in areas such: as data portability; the Internet; and distributed computing. They also complicate the task of integrating other tools into the mathematical programming process.

The application of the Internet and distributed computing to the area of optimization is still in its infancy. The most notable attempt to date is the Kestrel solver from the NEOS Server [35]. OptML [23] also aims to leverage internet based protocols e.g. SOAP [43] in transmitting and solving optimization problems. *In order to leverage the power of the Internet in optimization, it is necessary to define a framework which not only deals comprehensively with the issue of model representation, but is also designed to enable distributed optimization over Internet based protocols.*

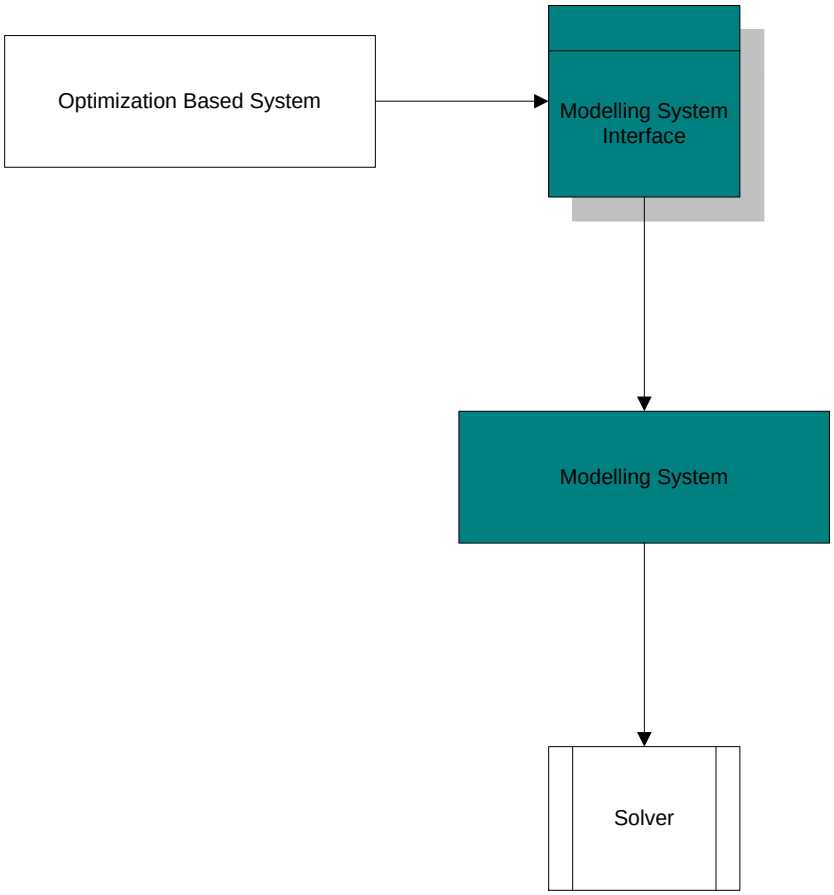

Figure 2.3: Hybrid implementation utilising an algebraic modelling system

The focus of optimization has shifted from obtaining a solution to other aspects of the mathematical programming process. These include post and pre optimality analysis, requirements analysis, model simplification, automated assistance for model formulation etc. As such, it is often necessary to employ additional tools during the mathematical programming process. A good example of such a tool is MProbe [6] which is used to perform pre-optimality analysis. However due to the proprietary nature of modelling systems or optimization model implementations, it is often difficult to integrate additional tools into the mathematical programming process. Any new representation format should strive to simplify this process.

2.3 New Initiatives

Low level input formats such as the MPS format have been subjected to independent revisions and extensions by a number of different parties. The result of these significant but unsynchronized efforts is that although these formats are very useful to many interested parties, they also suffer from many short-comings. Some of them originate in design assumptions which were more relevant at the time of conception than they are today, for example because the MPS format originates from the era of punch cards, it is still limited to 80 character records. Many other limitations come from the lack of a formal mechanism for amending these formats. The advent of the Internet, and advances in software technology have also highlighted additional shortcomings. Due to the significant demand for revamped and standardized formats for communicating optimization models, there are a number of initiatives aimed at achieving just this. However some researchers have chosen to aim for altogether new formats, with contemporary technical assumptions, taking advantage of the wealth of functionality and software available for exploiting the XML [31, 50] language, and designed from the ground up with the mathematical programming advances of the last 30

years, and expected for the next 30 years, in mind. The most well known of these are SNOML [27], and the OptML Toolkit.

Both of these projects have focused on a matrix level XML dialect for representing instances of linear and mixed integer programs. Except for a few differences both of them are fundamentally similar because they concentrate on representing the same level of information. At this stage the major differences lie in their support for non-linear and stochastic optimization, and representation of problem structure. Whereas the OptML Toolkit utilises a proprietary stack based language to represent non-linear problems, SNOML provides no specific support beyond the capabilities of MathML [29]. SNOML's support for stochastic programming is limited to support for multi-stage recourse and probabilistic constraints whereas OptML provides no specific support. In terms of problem structure OptML utilises statistical structure whereas SNOML uses re-usable blocks and stage coupling.

These projects focus on the lowest common denominator in model representation i.e. representing model instances/matrices. In essence, they provide support for representing information at the matrix level rather than the algebraic level. The advantage of this is that it may be easier to introduce a single matrix representation format or standard, than it is to introduce a single algebraic-level representation format. The drawbacks however include the loss of generality and the independence of models and model instances.

Despite the limitations of focusing on matrix level representation, these projects represent a very significant step towards solving the chronic problems associated with instance representation. In order to appreciate the contribution which these projects could make to the practice of mathematical programming, consider the problem of interfacing modelling languages and solvers. If there are M modeling languages and N solvers, an industry standard based either on OptML or SNOML would reduce the number of drivers required to interface these modeling languages and solvers to M + N. Without an industry standard M * N drivers are required for every modeling language to be compatible with every solver. This is of course discounting additional complications which different platforms, e.g. UNIX, Linux, and WIN32 introduce.

2.4 What Is A Comprehensive Solution?

This section attempts to provide the requirements for a comprehensive solution to the problem of model representation. It lists the desirable/ideal features of any solution to the problem of model representation. Although by no means exhaustive, it forms a good foundation on which to base a solution.

i. ***Generality:*** A representational scheme should be generic enough to encompass most of the optimization paradigms—e.g. linear programming, mixed integer programming etc, and constraint logic programming. This reduces the burden on application developers/programmers to cater for different representational formats for each problem variant or scenario. Moreover as with all other software requirements or specifications, models are likely to change or evolve, especially in decision support environments. It would be highly tedious and expensive to alter the model representation format and its associated interface each time this happens.
ii. ***Model and Instance Independence:*** Representational independence of general model structure and the detailed data needed to describe specific model instances is a prerequisite of any model representation scheme. It is quite likely that the same problem will be solved using different data sets on multiple occasions. A good example is a DSS which has at its core an optimization model. Such a system will ultimately generate different sets of model data and consequently new model instances depending on user

input. Arguably it is possible to regenerate the model each time, but this would actually require the use of custom code libraries, or representing the model instance with high level programming structures, both of which introduce their own set of problems. Apart from the inflexibility involved in such an approach, it potentially ties the implementation to a single toolkit and possibly vendor. It would also introduce a lot of tedium into the other aspects of optimization not directly related to obtaining a solution e.g. requirements analysis, pre and post optimality analysis etc. By not using a single representation format (*iii*) i.e. by implementing the model instance structure directly in code, it is quite likely that a number of bugs could and will be introduced into the system. The cost of such an implementation plus the added cost of debugging and maintaining it, provide a strong case for avoiding this where necessary.

iii. ***Single Representation Format:*** It should be a single representation format that is not only suitable for computer execution, but can also be transformed to formats suitable for other purposes e.g. mathematical use, managerial communication etc. This is related to item *(iv).*

iv. ***Support for Multiple Views:*** Added ability to support multiple views of mathematical models and model instances. It should facilitate model representation independent of modelling language syntax, solver specific input format, or solution algorithm. It should also be possible to transform content based on it to any other representation or format including modelling language or solver specific input formats using well defined scheme.

v. ***Vendor Independent:*** The solution should be vendor independent and ideally should be managed by a standards body or group. However it should be possible to develop vendor services and extensions e.g. solver or algebraic language interfaces, for it which can be plugged in using a well defined process.

vi. ***Portability:*** This not only refers to portability of information i.e. syntax definition files, models and model instances etc, but also to the portability of any associated binaries or code libraries. A number of optimization systems have become obsolete simply because they were tied to platforms which either became obsolete, unaffordable or infeasible to support. As such, the representational format should utilise industry standards such as XML to guarantee the portability of the data. Also any libraries or binaries associated with the format should be implemented using a platform independent language such as Java.

vii. ***Delineation of Responsibilities:*** There should be (and there is) a clear delineation of responsibility of the software involved in the optimization process and the solution should take this into account and reinforce it. For example, model representation is the sole responsibility of a modelling system, whereas obtaining a solution is the responsibility of solution algorithm or solver. By utilising a common method for problem representation, optimization software do not have to rely on explicit programmatic interfaces, and as such different software can be combined during the optimization process, without undue concern about interaction between them. Such delineation becomes even more important if the aim is to eventually publish and deliver optimization software via the Internet, as there will obviously be a need to access a variety of services independent of input formats.

viii. ***Support for Distributed Optimization:*** Considering the wealth of research into distributed computing including more recent work on grid computing, and the limitless opportunities which this area provides to the practice of operations research (see ix), a representational format should aim to cater for it. In the view of the authors, this is best done with a representational framework which at least meets criteria *(i-vii, x).*

ix. ***Exploit the Power of the Internet:*** The Internet represents the largest open medium not only for transmitting information, but also for delivering applications. The latter of which makes it possible to view it as a single distributed computer of near unparalleled power. Internet based protocols such as SOAP, and standards such as WSDL [46] and UDDI [47] have made it possible to access and invoke applications in standard manner, the only requirement being the ability to connect to the Internet. This ability to deliver

and access computer software via the Internet using simple Internet based protocols with relative ease and simplicity has opened up opportunities which hitherto had been unknown or only dreamt off. Of major importance is the ability to deliver optimization software and or decision support software via the Internet [3,4,5]. In order to take advantage of these opportunities a consistent and robust means is required not only to represent optimization model instances, or models, but which also makes it possible to define or publish these software interfaces in a consistent manner e.g. WSDL. This demands that such a solution not only be generic but also have the widest possible range of applications as possible. As opposed to say for example just representing model instances, it should also cater for models, analysis, documentation etc. Section 3 provides additional details on the opportunities which the Internet provides to OR practitioners.

x. ***Accessibility:*** The components which comprise the solution should be openly and freely available. This not only encourages faster uptake, but also greater participation from other members of the operations research community.

3 The Internet: Opportunities for OR

The Internet is the largest free and open repository of information known to man. Hence the primary and probably the most obvious opportunity, which it affords to OR is the ability to easily, share or disseminate information. The importance of this to both the research community and the commercial world cannot be overstated. However information dissemination and access do not constitute the greatest opportunity which the Internet provides to OR practitioners. In fact easy access to information, and ease of publication is associated with the Internet and applies to every field of endeavour including OR, and as such can be taken as a given fact. Hence it is not worth considering in any great detail within the context of this paper.

The other opportunities that the Internet offers to OR only become clear when we fundamentally alter our perception of it to fit two models: (a) a single distributed computer; (b) an electronic marketplace for optimisation resources. To appreciate the relevance of these views we must consider the problems associated with utilising optimisation based approaches for solving real world problems. This is best illustrated with an example. Consider the case of *Eisengard Sharedealing Inc.* Eisengard operates in a very competitive commercial environment. Established in 1898, up until the early 1990s it enjoyed a lucrative position as a market leader in the private investor brokerage market. This was of course before the web led to online sharedealing services, opening up the competition floodgates. Eisengard has recently launched its online share dealing service; however it still finds itself lagging behind its newer, smaller and more innovative competitors. To compound matters it is suffering from a severe downturn in trading volumes brought about by harsh stock market conditions. Long term investors are dissuaded from holding stock market instruments due to low and sometimes negative returns. Short to medium term investors are dissuaded from trading by the volatility of stock indexes. A management group is championing the idea of introducing an online portfolio manager (a DSS) to its range of services. This would provide portfolio-planning services for long term investors, and financial engineering (derivatives etc) to enable short-term investors exploit the volatility in stock markets.

In order to achieve this, the Eisengard board has assembled a working group in order to evaluate the feasibility of this idea, and prepare an initial proposal highlighting the risks associated with implementing it. The vision which this group has of the proposed system is that of a web front-end, utilising data which Eisengard currently maintains on its enterprise database, and driven by a shared optimization engine.

Up until this point this has been a non-standard problem scenario in the sense that the motivation for introducing an optimization based solution differs from one industry to another, and quite possibly from one organisation to another. However the results of the evaluation will be much more standard in the sense that the problems associated with utilising optimization techniques to tackle real world issues transcend industry, sector and business borders, and even the commercial vs. academic barrier. The Eisengard survey is likely to highlight one or more of the following problems:

Awareness: The working group may not be aware of the relevant optimization approaches that can be applied to its problem. In its report it is likely to request that consultants be engaged in order to recommend the best optimization paradigm and/or model(s) to suit its requirements. Even after the best possible optimization approach or paradigm is chosen, the working group also has to decide the best possible means of implementing the model rapidly. If the organization lacks OR knowledge or has never implemented a DSS, it is also likely to require the services of consultants in order to draw up a list of implementation options.

Accessibility: Most organisations considering a DSS solution do not have access to or own copies of the decision technology components required for a successful implementation e.g. models, modelling environment and possibly model data. As such, organizations have to purchase the individual components required after identifying the right options. This identification process would probably involve evaluating solutions or products from a variety of vendors in order to find the option that best matches the organisation's needs. As opposed to mass marketed software, the cost of this evaluation process can be quite high especially if the organisation in question does not have any in-house optimization knowledge. In which case, it would probably have to resort once again to outsourcing this task to external consultants.

Compatibility: Most organisations have a well defined IT infrastructure, and additions to this infrastructure have to meet specific requirements. Referring to the example scenario, if the Eisengard infrastructure is based on the SOLARIS platform, then all new software would have to be able to run on this platform. Requirements such as these only serve to narrow down the list of options available to an organisation, and leads to decisions based on the convenience of the solution rather than the features of the product in question. For example, it is quite likely Eisengard will discount the use of a very powerful solver tailored to financial optimization models, in favour of a more generic version simply because the latter can run on their operating system.

Applicability: Majority of optimization technologies are marketed in a generic version which may require further customization to suite the needs of individual organizations. For example, if an organization purchases an algebraic modelling environment, it may discover that it has to write a specific solver to meet its needs as opposed to using one of the generic solvers bundled with the system. In fact it is also possible that an organisation does not find any mass market solutions which meet its needs and has to resort to creating a custom implementation from scratch. This is not only an expensive, but also a time consuming process which could quite easily result in the duplication of effort especially if the software has already been implemented in a research capacity but left unpublished due to the costs involved in distributing it.

Interoperability: Many real world problems require the combination of multiple technologies to provide a satisfactory solution. In our example scenario, Eisengard would probably require to combine web technologies such as HTML, Java, one or more solvers, and analysis tools in order to implement its solution. Moreover the company would like to leverage its existing database, for example information already held on stock prices, customer portfolios, risk profiles etc, as opposed to recreating this from scratch simply for the purpose of implementing a DSS. Considering that each technology has its specific input formats,

language, and other idiosyncrasies, combining them is a lot easier said than done. In some cases such a level of integration may simply not be possible especially where there is a large age difference between technologies, for example when integrating to legacy applications. In cases where it is possible, it requires a great deal of system development effort, which is not only time consuming but can also be expensive.

The Eisengard working group will probably highlight the need to overcome these obstacles in its feasibility study, and the potential (possibly unknown) cost associated with such a project. The results of the survey may dissuade the board from authorising the project, especially if it deems that the risks and associated costs of the project far outweigh any benefits it may add to its market position, especially where these benefits are not known or easily measurable.

It is not only the adoption of optimization technologies which is problematic; distributing optimization technologies or solutions can be also be fraught with difficulty. To illustrate with an example, consider the case of Craig a graduate student, who is working on an optimization model for valuing exotic financial instruments. He has developed a solution algorithm (solver) to cater for the special characteristics of instances of this problem. Craig would like to market both his model and if possible his solution algorithm to the investment banking market. When he considers this idea in detail he begins to realise that there are actually a number of uphill obstacles which he would have to overcome. These are almost symmetrical to the problems experienced by users of optimization technology.

Advertisement: As with all other products, consumers have to be made aware of new optimization technologies. With traditional products including mass-market software this is normally achieved through mass advertising campaigns. Given the specialist nature of optimization technology, this approach is often not cost effective or feasible. This is especially true in cases where the software or model is the product of an academic research exercise. Academic institutions and students often find it difficult to publicise their work, and the end result is that a lot of the work is lost or buried, except in cases where it has actually been commissioned by an external organisation. In our example scenario, Craig would find it very difficult to advertise his model or solver to the wider market place. It is quite likely that his research group may have links to one or more external organisations however they are hardly representative of the whole market place. This problem is symmetrical to the awareness and accessibility problems experienced by optimization technology users.

Heterogeneity: Similar to all other technology markets, there is a variety of platforms within the optimization market. This often means that suppliers have to support a number of different platforms. This is compounded by the fact that a number of optimization software still utilise platform dependent programming languages such as C/C++, as opposed to more modern platform independent languages such as Java. Needless to say the cost of developing for and supporting multiple platforms can be quite high, and choosing to develop on a single platform obviously narrows the appeal of the product. As such, product providers are left in an almost impossible solution where they have to develop and support products on a number of platforms. This problem is also compounded if the software in question is a plug-in for other software or requires other software to work. For example if the product is a modelling environment which requires solvers to function, the product provider not only has to contend with operating system specific issues, but also with different solver interface mechanisms. In our example scenario, Craig the graduate student has to decide not only on which operating system to develop for, but also has to ensure that the interface to his solver is as generic as possible. This problem is symmetrical to the compatibility problems experienced by consumers.

Versioning: Recent experience dictates that computing technology is an ever changing phenomenon. It is not possible to rely on the stability of technology products or platforms. This is illustrated by the constant releases of operating system versions and related patches.

Suppliers of optimization technology are often forced to play catch-up with operating system, or platform vendors. Occasionally perfectly stable and working software is rendered useless due to changes in the underlying technology e.g. mass migration to a newer operating system. The cost of this is often high and occasionally is plain and simply impossible to bear. A number of optimization software has become obsolete simply because the platform to which they were originally ported is no longer in existence or has simply become uneconomic to support. Even in the absence of shits in the user platform, a variant of this problem is encountered if there is a need to upgrade and maintain the software over time. This problem is the dual of the applicability and interoperability problems experienced by optimization technology consumers.

Customization: This problem is symmetrical to the applicability and interoperability problems experienced by consumers. In the same way in which consumers have to integrate optimization technology into their existing infrastructure, or with other pieces of optimization/decision support technology, providers are faced with the problem of offering coordinated or integrated interoperable software solutions. Given the high cost of making customized software for each scenario, producing near generic solutions is often the most feasible way forward. However achieving a near generic solution is often a difficult task especially if the software requires input from or provides output to other software i.e. is used within a suite. This problem is compounded if the other components of the suite are produced by different vendors or providers. Even in the case where there is no need to produce an interoperable solution, consumers often request customizations of technology which fit their particular problem scenario. Needless to say, the cost of producing such solutions can be quite high.

The problems encountered by both consumers and providers of decision technology software are caused to a great extent, by the model used to distribute optimization software. Optimization and on a wider level operations research is a specialist market, and as such is not suitable to the traditional models used to distribute software. It is uneconomical for the software providers, and as such there is often minimal or zero marketing of optimization technologies e.g. in the case of solutions produced as part of a research effort. In the case of software which does gain a foothold in the market, it is often expensive to keep the software up to date and to cater for the needs of all the consumer groups. On the part of the consumers, there are a variety of products and solutions out there, majority of which they are not aware of. In fact it is probably safe to say that a number of organisations do not realise that the problems which they face can be solved by the use of optimization technology. In the case where the companies are aware of the applicability of optimization technology to their problem, they are often unaware of all the alternatives available to them. Even when all the obstacles are overcome, the organizations have to integrate the solutions into their existing technology infrastructure. This is of course assuming that the organization in question is willing and able to bear the cost of the exercise to this stage and beyond.

These problems can be overcome or at least mitigated by leveraging the capabilities of the Internet, and advances in Internet based software technology particularly in the sphere of web services and grid computing. To achieve this, we must alter our perception of the Internet to that of a single distributed computer or that of an electronic marketplace for optimization resources.

3.1 The Internet as a Distributed Computer

Advances in distributed computing technology, and also grid computing have enabled application developers and architects to treat networks (including the Internet), as a single computer. Distributed computing standards such as WSDL, SOAP and UDDI have enabled the delivery of logic over the Internet or any other medium which supports Internet based

protocols such as HTTP, HTTPS, and SMTP. In this paper we are predominantly concerned with the ability to deliver optimization functionality over the Internet, as this ability circumvents a number of the issues involved in the use and distribution of optimization technology.

In order to effectively deliver optimization technology over the Internet, there are three major problems that need to be addressed namely: (a) description of optimization functionality; (b) registration of optimization technology; and (c) a means of invoking optimization software over the Internet. The following paragraphs describes the technologies and standards which can be used to address these problems, and also highlights the building blocks that would need to be put in place for the technologies to be applied on a wide scale.

Describing Optimization Technology: WSDL provides a standard means of describing software delivered over the Internet, particularly interfaces to such software and any additional meta-data needed to access the software. Therefore, it can be used to describe the interfaces to optimization technology. This could solve the *interoperability* problems which consumers experience, but for the fact that optimization technologies have vendor specific input requirements. There is no portable means of describing optimization resources e.g. models, model data and solutions, however if such a means did exist, then it would be possible to solve the *interoperability* problem experienced by users, and to an extent the *customization* problem experienced by suppliers. By utilising a portable and abstract means to describe optimization resources, it would be possible to integrate different technologies from different vendors with little or no effort. WSDL meta-data could be used to provide further information such as the quality of the software e.g. the quality of solution algorithms or the applicability to particular problems. The use of WSDL meta-data could be used to address the issues associated with *applicability*. By abstracting the interface to optimization software, thereby making it easier for various vendors to publish their software, it would be possible to have different flavours of the same software concept e.g. different solvers that cater for specific scenarios which share the same interface but are differentiated by meta-data.

Registration of Optimization Technology: As already mentioned UDDI provides a means of advertising, discovering, and integrating web services. It provides a 'yellow' and 'white' pages type of service for software vendors and consumers. By utilising the UDDI standard, optimization software vendors could publish their software to a central repository of optimization software, i.e. a type of '*Decision Support Central'*, where vendors can advertise their products, and consumers can search for software. This simultaneously solves the *advertisement* and *awareness* problems experienced by vendors and users respectively. Vendors can advertise their software (possibly free of charge) to a central repository where users can search for the software. The use of UDDI is of course contingent on the ability to describe the interfaces and characteristics of the software being advertised using a standard such as WSDL. This in turn is also contingent on being able to abstract away the vendor specific interface details of the software.

Invoking Optimization Software: The SOAP protocol provides a standard means of invoking software utilising Internet based protocols such as HTTP and HTTPS. It is based primarily on the exchange of XML documents and attachments. In a scenario where it is possible to abstract the interfaces to optimization software using XML, then it is easy to envisage a scheme where optimization software can be located on a network and invoked using SOAP. Software vendors could locate their software on the Internet, and publish the interface using WSDL, and users could invoke the software using SOAP. This solves the problems of *heterogeneity* and *versioning*. Vendors no longer have to concern themselves with details of the user's environment, and can develop for a single environment i.e. their server environment. In the case where the software is intended to form part of a wider suite of technologies, provided that the interfaces to the other components of the suite can also be abstracted using a common XML terminology, then the vendor does not need to concern itself

with interconnectivity issues. Users can mix and match products based on their needs, especially if all of the products in question conform to the same interface conventions. The problem of *versioning* looses relevance, as changes to the software are completely transparent to the users, excluding changes to the core interface. Even in the case of changes to the interface, the vendor can use a phased roll-out of a new interface i.e. keep supporting the old interface until all their customers have moved to the new interface. In the scenario where the vendor chooses to update or fix the internal workings of the software, or change its platform, the changes can be made without the knowledge of the users. This is because the software is centralised and is accessed over the Internet utilising a straightforward protocol, and is not located or tied to the user's environment. This mode of delivery is not only cheaper but also means that vendors have much shorter release and bug fix cycles. The use of SOAP will also solve the *compatibility* and *accessibility* problems experienced by users. Users will no longer require that optimization software is compatible with their existing infrastructure, as there will be no need to locate the software on their IT landscape. Provided the software is accessible via SOAP it will be easy to integrate it with their existing systems.

By utilising web services standards such as WSDL, UDDI, and SOAP, it is possible to deliver optimization functionality over the Internet. Given the specialist nature of the market for optimization technology, this presents the most efficient means of utilising and distributing optimization technology. However the optimization market is fragmented due to the prevalence of vendor standards, and in order to achieve a situation where optimization software can be effectively delivered over the Internet, there is a need to abstract away from these standards. This is best achieved using a meta-language such as XML. This can be used to abstract vendor specific interface requirements, so that software can be accessed/invoked in a reasonably straightforward manner. This doesn't imply that vendors cannot publish and distribute their software using their own XML based interface standards; however this will only be a slight improvement if at all to the current situation. Users would still have to contend with the same *interoperability, compatibility,* and *applicability* which they face at present. Conversely vendors still have to face the *heterogeneity* and *customization* problems which they face at the moment.

Another approach to delivering optimization software over the Internet is the *ASP (Application Service Provider)* model. An application service provider (*ASP*) is a company that offers individuals or enterprises access over the Internet to applications and related services that would otherwise have to be located in their own personal or enterprise computers. Sometimes referred to as "apps-on-tap," ASP services are expected to become an important alternative, not only for smaller companies with low budgets for information technology, but also for larger companies as a form of outsourcing. The OSP [40] project utilizes this approach for delivering optimization software over the Internet. Whereas this approach is a viable alternative to the traditional models used to distribute optimization software, it doesn't solve the full range of problems that have been identified above. It probably solves user problems associated with *awareness* and *accessibility,* and the *advertisement* and *versioning* problems experienced by vendors. However the other problems remain unabated with this model. This becomes clearer if we consider the fact that optimization technology is rarely used in a standalone fashion excluding perhaps applications within the research arena. For example, this model is not applicable in the case of a company which plans on implementing a decision support system. However in the case of a single or a handful or individuals who wish to make use of an algebraic modeling environment, then this may offer the most cost efficient alternative.

3.2 The Internet as an Electronic Marketplace

The previous section covered the possibility of delivering optimization software over the Internet, and provided details of the technological building blocks that can be used or would

be required to achieve such a situation. From the previous section it is quite clear to see how the Internet could function as a common medium where software vendors and buyers can conduct business. As such this section will not cover these details again as this will be redundant. The only concept not addressed by the previous section is that of distributing re-usable optimization resources. This includes among other things models, model data, analysis hints—e.g. query patterns used to identify a stable solution, etc.

Among the issues involved in using and selling optimization resources are those of *awareness* and *advertising*. Users are often not aware of optimization techniques or models which are applicable to their particular problem, even in the case where the problem is a relatively standard one e.g. mean-variance portfolio optimization. In the case where the users are aware of the optimization paradigm that can best be applied to their problem, they face the possibility of having to implement models from scratch. They may also have to provide or collect the data for the models, which in some cases may be wholly unnecessary. Sellers may also posses models which they wish to publish or sell. One option for doing this is of course selling the mathematical formulation of the problem, however buyers are faced with the task of having to implement this once they obtain it. The sellers are also faced with the challenge of advertising the model in its current format. In the case where the model is the product of academic research, the only exposure it is likely to receive is in a scientific journal.

If we consider a scenario where it is possible to describe a model implementation or any other optimization resource using an abstract notation, say for example XML. A pleasant side-effect of such a notation is that these resources will ultimately become re-usable entities. This could lead to a situation where they are marketed or distributed as commodities in themselves. It is also possible to establish a common electronic brokerage service where buyers can purchase resources and easily integrate them into their environments with little or no programming effort, and sellers could sell resources provided that the resources are represented using a common notation or lingua-franca. By utilising a common notation, they can ensure that their products can be utilised in the widest variety of systems/environments possible. Moreover for electronic advertising purposes, it is easier to cater for a scenario where all resources are described using a common, user friendly and publishable notation.

The Internet can thus be used as an electronic marketplace where re-usable optimization resources can be traded. In addition to the resources traded on this marketplace, it could also be used to provide training material to aid novice users choose the best optimization resources which best apply to their problem scenario. The market could also provide additional data on the resources such as applicability, quality, tolerance, testimonials etc.

4 Framework Components

This framework was the product of a decision support system project at London's Imperial College. At the onset of the project it was clear that there was a need to support a variety of optimization paradigms e.g. linear programming, stochastic programming etc. and even constraint programming. In a decision support environment particularly one in which optimization or constraint programming models are generated at runtime using heuristics or rules, it is important to provide support for a variety of paradigms. It therefore was apparent that an abstract and portable means was required to represent models, irrespective of the model type i.e. linear, non-linear etc. and irrespective of the underlying technology that would be employed to process the model. As opposed to using a proprietary standard or format, a decision was made to use a meta-language such as XML. The *Algebraic Markup Language (AML)* was created for this purpose. It is a markup language based on the XML language, and allows the portable representation of models. It is flexible enough not just to support most if not all classes of optimization models but also constraint programming models.

Given the need to differentiate between models and model instances, it was also necessary to abstract the representation of model data. It is quite feasible that a model could be solved repeatedly with a wide range of data sets, and this is indeed done during analysis. As such it would be wholly inefficient to tie the model data to the model itself. Also different solution subroutines require data input in different ways, hence the data format in itself has to be abstracted to enable the easy merging of the data and the model to create an instance for a particular solution algorithm. The *AML* syntax caters for the separation of model and model data, and provides constructs for the full representation of model data.

A model and its associated instance(s) would only be artefacts of modern art if they served no purpose. Their purpose is generally realised when they are solved by invoking a solution subroutine, or when they are analysed. Leaving the complexities involved in invoking solvers, libraries or analysis components aside, once these software are invoked the data they return generally has to be utilised either by a decision support system, or any other tool or software responsible for presenting the results. In the same way in which there is a need to abstract the representation of the models and instance data, it is equally important to abstract the representation of solution, infeasibility, and analysis data. Various optimization systems e.g. solvers, analysis tools have vendor specific formats for representing their output and given the necessity to abstract away from vendor specific formats and thus reduce vendor dependencies, aid usability and integration, a requirement emerged for an abstract representation for reporting data i.e. solution, infeasibility and analysis information. In order to achieve this, the *Optimization Reporting Markup Language (ORML)* was devised. It is an XML based markup language which enables the representation of solution, infeasibility, and analysis information. AML and ORML form the basic building blocks of the OOF, and as such can be used in isolation of the other components of the framework.

Having defined abstract representations for models and instance data, there was a need to define a policy for converting these representations into a concrete model instance or model representation for a target modelling system, solver or analysis software, and also for converting the results of these target systems into the reporting data representation format. Ultimately to obtain a solution to or analyse a model, the model and its data have to be converted to a format which the target system or library can understand. In the same guise the format utilised by the target system for representing its output has to be converted to the *ORML* format so that it can be utilised by the calling application. In the case where there are N target systems, this has to be done N times, because the model and data representations are abstract, and the output from the target solvers or analysis systems are vendor specific. Although it is not possible to avoid this situation completely, it is possible to abstract the conversion process in itself. This is achieved by the *Optimization Service Connectivity Protocol (OSCP)* which is a Java library that abstracts away from, and shields the application from the complexities involved in integrating external components. This library is similar to the JDBC API provided by Sun Microsystems. In essence, the low level protocol details involved in communicating with the third party component is delegated to a *driver*—a software library that is responsible for low level interfacing to a third party component or library. Although the driver can be provided by the component vendor, it doesn't necessarily have to be. It can be written and distributed by anyone, or organization, therefore there doesn't have to be any ties to the product vendor.

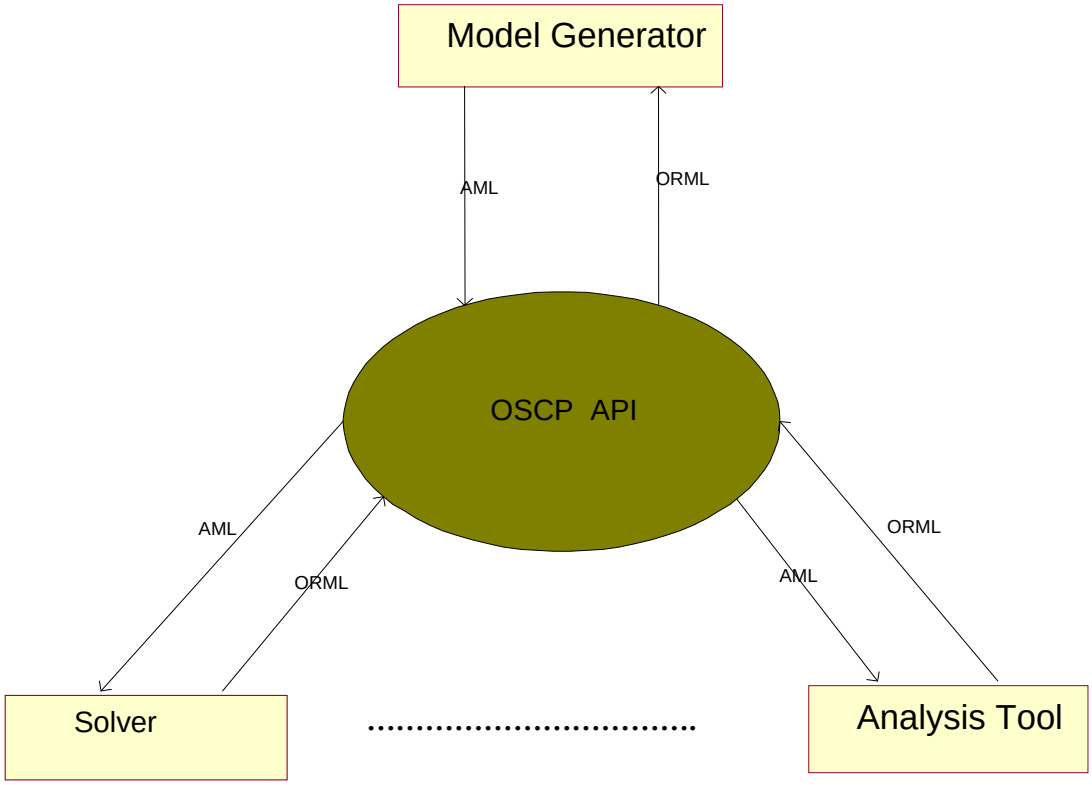

Figure 4.1: OSCP API abstraction of software interfaces

The library abstracts the process of integrating third party products such as solvers and analysis tools, and simplifies or abstracts the protocol involved in communicating with such products. The reason for implementing the *OSCP API* in the Java language is to enable use across multiple platforms. This avoids the situation where it is tied to a particular operating system or environment. The API was developed for a scenario in which there is a need to integrate optimization software running within the same environment i.e. local integration (illustrated by Figure 4.1), as opposed to a scenario where the software being integrated runs in a remote location. The *OSCP API* in its present state is still open to discussion, and is likely to be revised in the near future so as to take into account contributions by vendors and industry experts.

The model in which optimization software is purchased and installed on the users machine, or on a more specific note, one in which decision support software is bundled with optimization software and installed on the users machine is quite an expensive and administration intensive model for distributing software. This applies to all software in general and isn't limited to decision support software, hence companies often opt for the model where software or functionality is held in a single repository and distributed or accessed via Internet based protocols.

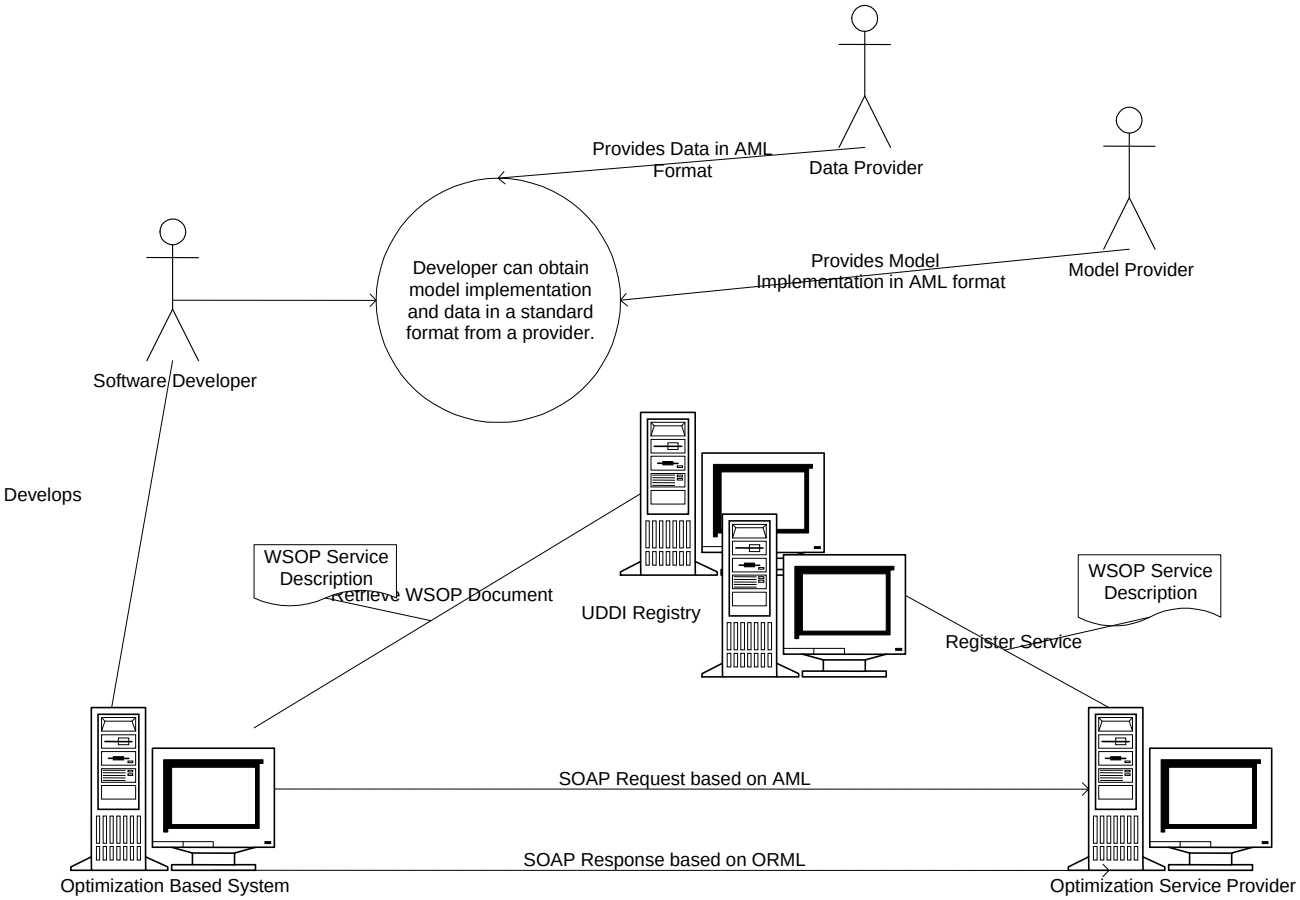

Figure 4.2: Framework web services approach to OR software delivery

This model is beginning to gain prevalence in enterprise environments, as web browsers and Internet protocols have proved to be a reliable medium for running or accessing software. Technologies such as Java Applets have made it possible to download software from a central repository and execute on the user's browser. Other technologies such as Java Servlets, Java Server Pages, XML, and SOAP have made it possible to access software remotely using Internet based protocols. Consider a case where an organization wishes to distribute a decision support system which requires one or more solvers and analysis tools. The organisation would face a choice between three major options: (a) create an application which has to be installed on the user's machine together with all the software it depends on i.e. solvers, analysis tools etc.; (b) create an application which is installed on the users machine but uses distributed computing technologies e.g. SOAP, RMI, EJB etc. to access other components installed in a central repository; (c) create a web application e.g. a Java Applet which uses distributed computing technologies to access central resources. Of all these options (c) is the most cost effective, followed by (b) for reasons which will be explained later on in this paper. This realisation is beginning to dawn on OR practitioners hence the various projects aimed at developing remote and centralised services for delivering optimization and decision support software e.g. NEOS, and DecisionNet [10]. The authors recognised this need as well, and as such one of the core aims of the framework is to leverage the power of the Internet and Internet based protocols so as to enable delivery of optimization functionality over the Internet. In fact, one of the major reasons why XML was used as the basis for *ORML* and *AML* was to open up the possibility of taking advantage of Internet messaging protocols such as SOAP. The *Web Services Optimization Protocol (WSOP)* which forms part of the framework was created in response to the need to distribute functionality using Internet based protocols. Its role in the framework is illustrated by figure 4.2. It attempts to address the common issues which arise from distributing and accessing OR software over the Internet, and recommends in terms of WSDL the basic interfaces for services delivered over the

Internet. It is not intended as a standard, but rather a recommendation contributed to by members of the research community as well as product vendors.

The four components of the framework *(AML, ORML, OSCP,* and *WSOP)* between them provide an integrated solution to the many problems facing delivery and use of optimization software in today's environment. This is achieved by first of all providing a solution to model, model instance, and reporting information representation, and then building on top of these to alleviate the problems associated with integrating optimization software, delivering and accessing optimization functionality over the Internet. The features of the individual components of the framework are covered in the following subsections; however the general strengths of the framework can be summarised as follows:

Portability: It is not tied to any particular vendor or standard and it is portable across different platforms and systems. Its constituent markup languages abstract away from vendor specific requirements and formats, and the use of XML guarantees portability across applications and platforms. This is because the use of XML effectively delegates the interpretation of the data to vendor specific plug-ins or to the back-end of optimization services delivered over the Internet. XML in itself is pure text and as such can be utilised across the majority of platforms and can be transmitted over basic protocols. In addition to the use or XML, the choice of Java as the implementation language of the *OSCP API* also guarantees the portability of code libraries. Finally, by basing the *WSOP* on web services standards, the choice of the implementation language can be delegated to the service provider and is completely invisible to the service user or the service registry. In fact, the only thing which all three parties share in common is their dependence upon the Internet. Both the *OSCP API* and *WSOP* recommendation both abstract away from vendor interfaces and vendor implementation details, therefore enhancing the portability of the framework.

Delineation of Software Responsibilities: It is almost a given that different software are employed at any one time when implementing a decision support system or applying optimization theory to a real world problem. Although these software perform different roles it is often difficult to establish a clear demarcation between their responsibilities therefore complicating the integration process. For example it is difficult to differentiate solvers embedded in spreadsheet applications from the application itself; as such developers wishing to exploit these solvers are forced to use the full blown spreadsheet application in its entirety. To present another example, an algebraic modelling environment is of little practical use without its stable of solvers, and as such using the environment purely for model representation, leads to the redundancy of the embedded solvers. In an ideal world developers should be able to integrate a variety of software in order to implement or arrive at a solution. For example it should be possible to use a representation format, with a variety of solvers possibly from different vendors, one or more analysis tools, and one or more model presentation tools without being forced to integrate the analysis tool into the modelling language which provides presentation and representation, or to utilise the solvers provided with either the analysis tool or the modelling language. The framework enforces the delineation of software responsibilities, by utilising abstract representations for information (models, instance data etc.), and by defining abstractions of software interfaces in terms of the role of the software. For example a solver interface is abstracted in a different way from that of an analysis tool. This is because the primary role of a solver is to obtain a solution where one exists, and an analysis tool is responsible primarily for analysis. As such, the interfaces defined are respectively for obtaining a solution and performing analysis. In essence, it enables developers to assemble software which fulfil various responsibilities in order to achieve the desired solution. For example, based on the framework, a piece of software can be used for formulation, another for presentation, a solution algorithm for obtaining a solution, and an analysis tool for analysing the results without having to worry about the integration of these different sets of software. This is because they all operate on abstract representations

defined by the *AML* and *ORML* languages, and can be easily integrated with either the *OSCP* or *WSOP* recommendation.

Generality: The framework is intended as a generic solution that supports the majority if not all of the optimization paradigms currently in existence, in addition to constraint programming. As already mentioned, the framework maintains a clear delineation of responsibilities between its components, and the responsibility of information representation lie with *AML* and *ORML*. As with any other situation where there is specialization, this provides unique advantages. By focusing solely on representation, and delegating the processing and manipulation of the representation to other components, it is possible to specify a representational scheme that is robust enough to meet the demands imposed by the various optimization paradigms and approaches. An added bonus is also the ability to represent constraint programming problems.

Ease of Integration: One of the core aims of the framework is to facilitate the easy use and integration of optimization software. In order to achieve this, it abstracts the interfaces of optimization software, and shields the calling applications from the low level interface intricacies. It offers two main modes of integration, either using the *OSCP* or *WSOP* recommendation, neither of which exposes the calling application to the low level details involved in communicating with the software. The *OSCP API* utilises abstract interfaces for communicating with third party software, and delegates low level interface invocations to an implementation of the *driver*, and as such shields the user from low level integration details. The *WSOP* recommendation on the other hand specifies services in terms of WSDL, and relies only on the user being able to send and receive information over the Internet. The details of the services or their implementation are completely shielded by the WSDL descriptions and the Internet.

Internet Service Delivery: The framework enables the delivery of optimization services over the Internet. This is achieved in great part by the use of XML as a representational format. This enables transmission of optimization commands/requests over Internet based XML messaging protocols complete with model, or/and instance data, and receipt of responses in the form of solution data, infeasibility and analysis information. In addition to the ability to utilise Internet messaging protocols, the use of XML enables the specification of services or interfaces which are common to optimization software delivered via the Internet. This is achieved via the *WSOP* recommendation. This is provided as a tool to enable service providers to deliver software functionality via the Internet, and also reduce the overhead involved in integrating optimization services, thereby encouraging users to adopt this model for purchasing software.

Re-Usability: The framework is organized using a component based approach in a manner which reduces the coupling between the components wherever possible. The integration components *(OSCP* and *WSOP)* are built on top of the representational components *(AML* and *ORML)* and as such have to be used in conjunction with them. However, the representational components can be used standalone, and the two integration components have no dependencies on each other. In essence, although there is a high level of cohesion between components, there is very minimal coupling except where explicitly necessary. This model makes it easier to re-use individual components, seamlessly add new components, and extensively modify individual components of the framework with measured impact.

Although at its core the framework is concerned with the representation of information, it attempts to leverage its own strengths in this area to provide a recipe for integrating optimization software, and for delivering optimization functionality over the Internet. Hence the framework is described as an integrated solution which not only comprehensively solves the problem of model representation, but takes advantage of advances in computing technology particularly in distributed computing to aid in the delivery and integration of

optimization functionality. The following subsections provide descriptions of its individual components, including their intent and their core features.

4.1 Algebraic Markup Language (AML)

The Algebraic Markup Language or AML provides syntax for accurately representing optimization models, and model data. It's neither a new modelling language, nor does it attempt to define a new standard for optimization model instance representation. One only has to look at the current landscape of optimization software to become pessimistic about any such attempts. It is currently littered with a variety of modelling languages, industry and vendor specific input formats for optimization model instances. As such, introducing a new modelling language or input format would only serve to add to the current state of mayhem. *Rather what AML proposes to do is to abstract away from modelling languages and vendor specific formats completely. In essence this involves defining a representational format that can be transformed into a multiplicity of other formats.*

4.1.1 Background

Optimization models and model instances are ultimately just bits of information which can be encoded in a variety of ways. At the moment, this can be done with mathematical notation in an electronic document e.g. word document or rtf, with a modelling language supported by some modelling environment, or a low level input format for representing mathematical model instances. Representations based on mathematical notation are best reserved for communicating the idea of the model, and are ultimately unsuitable for computational purposes. At this point it is also worth noting that projects such as MathML do provide notation for representing mathematical notation as markup, however these are generally concerned with the presentation of the mathematical elements of the model, and although it is possible to define the mathematical form of optimization models in this way, it serves little purpose other than for presentation. MathML doesn't provide or cater for the concepts of computable models (matrices, model parameters, constraints etc.), or separate instance data; all of which are basic requirements of a representational format.

The presentation form of optimization models i.e. the mathematical form and any additional text required to explain it, is normally used to convey the idea of the model to other OR practitioners, computer scientists, management etc. This format in itself is barely useful for computational purposes, and as such has to be converted by an implementation process to one that is suitable for computer execution. This can be achieved by using an algebraic modelling system and language, or a high level programming language. Theoretically, low level input formats can also be scripted manually although this would be a highly tedious and error prone process.

In the case where optimization models are implemented using a high level algebraic modelling language, the implementation has to be compiled by the modelling system into a form which a target solver can recognize. This can easily be achieved using a software bridge. In essence one form of representation is converted into another form expected by a target component. In general modelling systems have a stable of solvers to cater for different varieties of models. As such, consider a case where a modelling system has M solvers. The process of bridging the modelling language to a solver specific representation has to be repeated M times. Consider the case where there are N modelling systems. This process in total would have to be repeated N x M times, as illustrated by figure 4.3. The problem is of course exacerbated by different operating systems or operating system versions. If there are Z target environments, then this figure could be anything as high Z x M x M. In the case where the model is implemented in a high level programming language such as C++, the process of interfacing to external solvers or modelling tools, the bridging/interfacing process still has to

be repeated. This is of course excluding the case where a solver is specifically implemented for, and integrated into the model.

Whether or not a model is implemented using an algebraic modelling language or a high level programming language, the intent of the implementation is purely for computation. More often than not, this is not suitable for any form of advanced or effective communication. To illustrate this point, consider an example where a management board has agreed on the requirements for an optimization model, chosen a suitable model, and implementation work has begun on the model. If at any stage during the implementation, problems are discovered with the model, and there is a need to modify it, the modification would have to be made to the mathematical notation of the model and any supporting documentation, and presented to the board for approval. If these changes are approved by the management, then they are fed back into the implementation process. The computational form of the model is completely excluded from the communication material as it will probably be unintelligible to managers, business analysts, and all others non-technical members of the decision making process. The opposite also happens when there is a shift in requirements. In this case it is likely that the original mathematical form of the model would be modified first in response to the new requirements. These changes then have to be applied separately to the implementation.

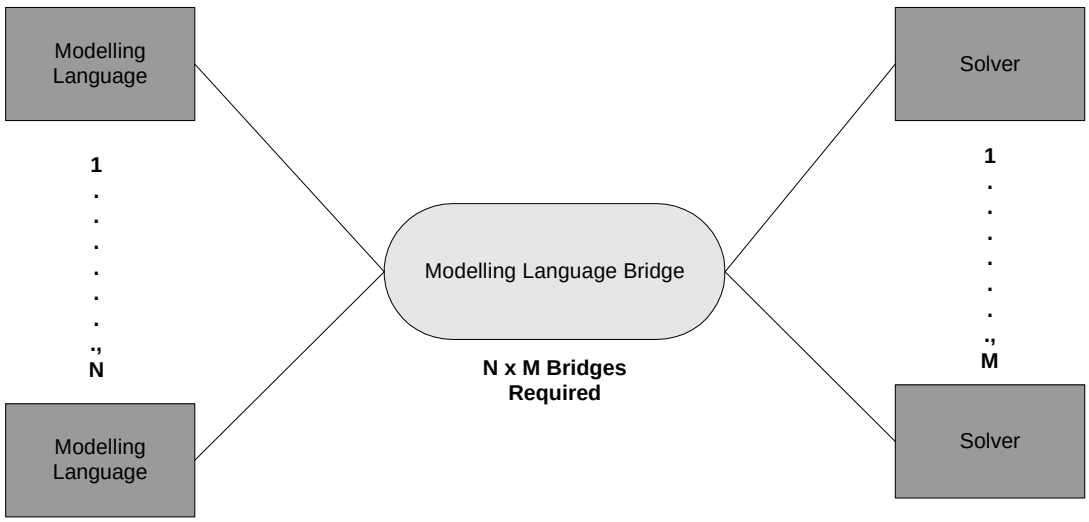

FIGURE 4.3: Bridging model representations and solver interfaces

It is clear that there are currently two main forms of encoding optimization models: (a) presentation; (b) computational. As the names imply, one is concerned solely with the presentation of models, while the other is concerned with a computable representation of the model, where *computable* refers to one that can be processed by a computer for the purposes of obtaining a solution or for analysis. These two schemes result because the computational form of the model is probably too large and complex or simply unintelligible to novices to be used as a communication tool. Whereas the presentation form of the model is purely for communication and cannot be used for computational purposes. Thus in a typical scenario, there are at least two disjoint representations of a model in existence, one for communication and the other for computation. Maintaining these two in sequence or transforming from one to the other can be quite an expensive process and often results in bugs being introduced to the model implementation.

In an ideal world, it should be possible to buy an off the shelf model implementation which has gone through the development and implementation cycle, and as such can be considered stable. However this introduces the added problem of having to integrate the model into an existing infrastructure, or having to obtain an environment where the model can be executed. Implementations are generally either tied to a particular modelling language, or operating

system, or even solver. Consider the scenario where a research student obtains an optimization model implemented using an algebraic modelling environment. If the student's university does not licence or support the modelling environment, the student may have to re-implement the model using a supported or licensed modelling environment. The same dilemma is faced by companies, and more so because they generally have a well defined IT infrastructure, and find it more difficult to integrate new products into it.

In general, the encoding schemes used for representing optimization models are too closely aligned with the ultimate target or use of the model. In essence as opposed to representing the information about the model, the schemes are more concerned with what the representation is going to be used for. As illustrated in the preceding paragraphs this leads to a number of problems, which can be summarised as follows:

i. ***Lack of portability:*** Models are generally not portable from one implementation platform to another. This could be as a result of vendor specific representation formats, or dependencies on a target platform or operating system. Ideally a representation format should not be platform or vendor dependent.
ii. ***Single purpose representation:*** Representations are used either solely for computation or presentation. Ideally there should be one representation format for both purposes.
iii. ***Loss of re-usability:*** The extent to which a model can be re-used is dictated by the language or format in which it was implemented. Users need to have an environment, platform or tool which supports the implementation language or format in order to effectively use the model.

Ignoring the intricacies of optimization models, it is possible to define an encoding scheme which abstracts away from the current forms of representation, and which can be used to generate multiple views. In essence, capture the information about the model without tying the representation either to an execution environment or to a particular view or use. This involves defining a modelling language or syntax which can be used to generate multiple views with little or no effort, and which more importantly than anything else is suitable for computer execution, i.e. as opposed to focusing on just capturing the information about the model for communication purposes it should also be suitable for computation purposes. It is highly unlikely that all the product vendors in the market today can be encouraged to adopt a single common format for representation, so it is important that the language is an abstraction of existing formats and doesn't explicitly or strictly require vendor support. Hence there should be no ties to any vendor specific requirements, standards or formats.

AML achieves this by using XML, and in particular the XSD recommendation. As already mentioned XML is a meta-language which can be used to define other languages. XML guarantees the portability of data by delegating the interpretation of the data to the receiver and/or sender. In essence the meaning of the data is decoupled from its representation. AML by building on the strength of XML provides a portable representation of optimization and constraint programming models. It serves as an abstraction of current representation formats, and supports multiple views generated either through the use of XSLT, custom programmatic constructs, or a combination of both. In essence, AML decouples the use of the optimization models from their representation. The use and interpretation of the model is ultimately delegated to the receiving application. It also guarantees the re-usability of models, and is not tied to any particular paradigm, modelling language, programming language, operating system, or vendor specific format. The following subsections provide more information on the advantages of AML, and a summary of its core syntax.

4.1.2 Advantages of AML

This section lists the advantages which AML offers over the current formats and methods for representing optimization and constraint programming models. They can be summarised as follows:

i. ***Portability:*** As already mentioned AML offers a portable means of representing optimization models. It is based on XML, hence it is text based and can be used across multiple platforms, environments and programming languages. It abstracts away from modelling languages and model representation formats by concentrating on the representation of the model and its associated data, and delegating the processing of the model to the target environment. Because the target environment is quite likely to require a specific format, AML supports the generation of other representation formats. In essence it is possible to generate other views of the model and its associated data from a single AML representation. This essentially abstracts away from vendor or platform specific requirements, thereby guaranteeing the true portability of models.

ii. ***Support for multiple views:*** AML supports the generation of different views of a model and its associated data from a single AML representation. There are a number of reasons why this is required, among them are: the need to support views specifically meant for communication; and views that are required for computation by a specific target environment e.g. an algebraic modelling system. There are a number of means by which this can be accomplished, among them: the use of XSL; and the use of a purpose built transformation engine. It may be possible to generate views of the model perhaps for communication purposes by the use of XSL stylesheets and a general purpose transformer. In the case of generating a representation for a specific algebraic modelling environment, it may be necessary to utilise a purpose built transformer designed and built specifically for the target environment. Figures 4.4 and 4.5 illustrate this.

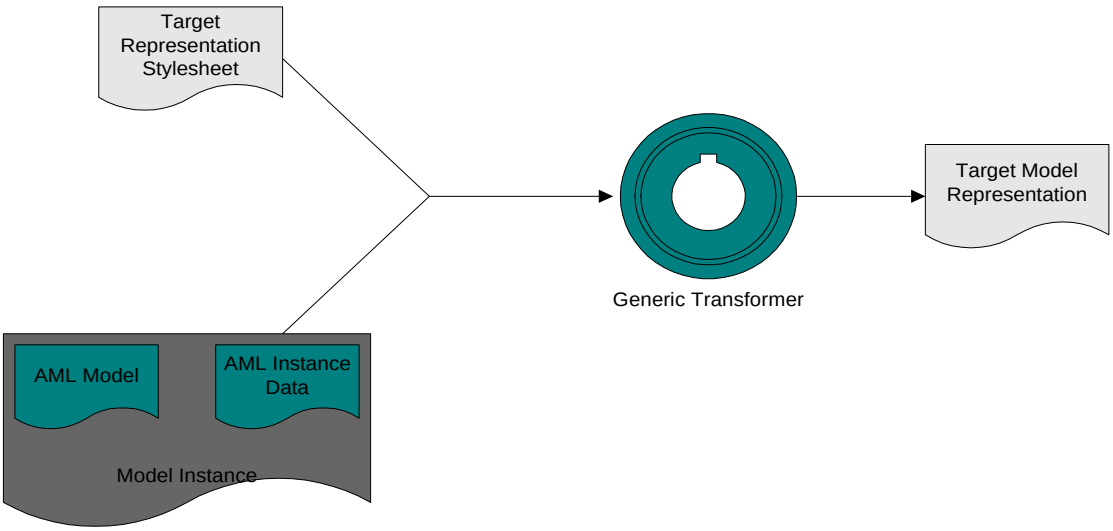

Figure 4.4: Translation using an XSL stylesheet

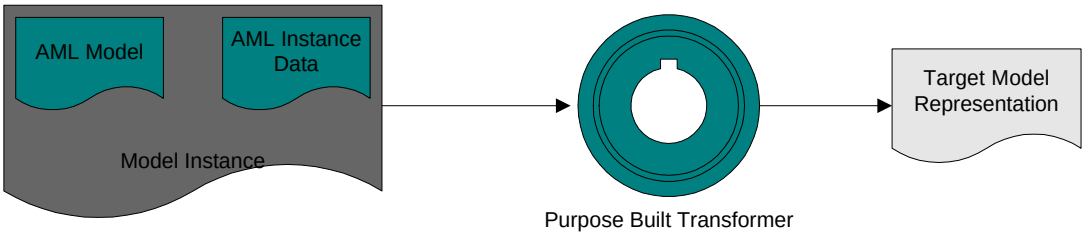

Figure 4.5: Translation using a purpose built transformer

iii. ***Model and instance separation:*** AML provides a clear demarcation between a model and the data required to create an instance of the model. Ultimately it will be necessary to solve a model repeatedly using different data sets. Therefore it is necessary to be able to specify the data set independently of the model. Else the model would have to be changed each time the data changes. AML enforces this separation of data and model, thereby ensuring that models can be re-used over and over again with different sets of data i.e. different instances of the model can be generated simply by specifying different data sets.

iv. ***Re-usability:*** AML guarantees the re-usability of models by two principal means: (a) by its portability; (b) by separating models and instance data. The portability of the AML format ensures that models can be easily ported from one environment to another. Therefore a model obtained in AML format can be easily ported to the user's local environment or an organisations local infrastructure. Also in a scenario where an organisation or a user uses AML as the format for representing optimization models, it is easier to change the underlying modelling infrastructure e.g. algebraic modelling languages, solvers etc. This is because AML does not have any vendor or format specific features and can be easily transformed into other formats. In addition to portability, AML guarantees re-use by separating models and the data required to represent specific instances. Therefore, models can be re-used with different data sets and it may even be possible to re-use data sets depending on the application. This not only makes it easier to perform tasks such as analysis, but fits into the decision support philosophy where models will are repeatedly run against different sets of data. Re-usability ultimately makes it possible to turn models into entities which can be distributed just like code libraries.

v. ***Generality:*** Unlike a number of other representation formats AML is not tied to any particular paradigm such as linear, non-linear, stochastic programming, or even optimization alone. It supports the representation of all known classes of optimization models, and the data required to represent their instances. In addition, it supports the representation of all known classes of constraint programming models and model data. All of this is achieved using a common syntax; hence there is no requirement to use a different syntax for different classes of problems or for representing constraint programming problems.

vi. ***Support for Internet based applications:*** Apart from providing a means for abstracting away from other languages or representation formats, one of the key benefits of XML is that is text based and can be used in conjunction with a number of Internet-based messaging protocols such as SOAP. By basing AML on XML, this feature of XML is automatically inherited by AML, and as such it is possible to use AML in Internet based applications. Models and model data can be transmitted using common Internet based protocols such as HTTP, HTTPS, SMTP, etc. As such, AML exploits and provides the means to exploit advances in computing technology in the areas of the Internet and distributed computing. The ability to transmit models and model data over the Internet or intranets completely re-defines the traditional model of distributing optimization functionality and to a wider extent the design and implementation of optimization based systems. It becomes possible to centralise functionality, and provide access to this functionality using a web services approach to software delivery. This has enormous

impact on the topology and cost of software systems. As opposed to the scenario where a system has to be delivered with all additional software such as solvers, analysis tools or even an embedded modelling environment, systems can be delivered as a skeletal structure which can then access additional resources via Internet based messaging protocols. This provides a whole host of advantages such as easier software delivery, ease of maintenance, and lower licensing costs.

4.1.3 Syntax Overview

This section provides a summary of the core syntax of AML. This is not intended as a definitive nor comprehensive list of the features of AML, but rather as an insight into the capabilities of the language and its core syntax. More detailed information on the AML syntax is provided in [12].

4.1.3.1 Documentation

AML is a self documenting language, and as such it is possible to embed documentation directly into models. There are two levels of documentation supported by the language: *onTextComment;* and *offTextComment.* The reason for this is so that it is possible to distinguish between code-level comments and user targeted documentation. The *onTextComment* documentation type is intended for user level comments i.e. comments that can be formatted and used to generate detailed reports for users, whereas *offTextComment* is used to document the model itself i.e. comment the model text. The reasons for providing model level comments vary, but normally centre on aiding easy understanding and improving maintainability. The following segment provides an example of an AML documentation element.

```
<documentation>
       <onTextComment>user level comment goes here</onTextComment>
       <offTextComment>model level comment goes here</offTextComment>
</documentation>
```

Figure 4.6: AML documentation fragment.

4.1.3.2 Operators

AML provides a number of operators including arithmetic, logical and set operators. These are typically used to build arithmetic expression and define model parameters, sets, constraints, objectives etc. Figure 4.7 provides an example of an expression definition which makes use of a simple division operator.

```
<complexExpression>
       <lhAlgebraicExpression>
              <operation>
                     <setOperation>SUM</setOperation>
                         <index>
                             <indexName>t</indexName>
                             <setName>securities</setName>
                         </index>
                     <bindingExpression>
                            <lhExpression>
                                   <elementReference>
                                          <attributeName>Dv</attributeName>
                                          <attributeIndex>
                                                 <indexString>t</indexString>
                                                 <setName>securities</setName>
                                          </attributeIndex>
                                   </elementReference>
                            </lhExpression>
                            <operator>*</operator>
                            <rhExpression>
                                   <elementReference>
                                          <attributeName>Dm</attributeName>
                                          <attributeIndex>
                                                 <indexString>t</indexString>
                                                 <setName>securities</setName>
                                          </attributeIndex>
                                   </elementReference>
                            </rhExpression>
                     </bindingExpression>
              </operation>
       </lhAlgebraicExpression>
       <operator>/</operator>
       <rhAlgebraicExpression>
              <expression>
                     <elementReference>
                            <attributeName>TimePeriod</attributeName>
                     </elementReference>
              </expression>
       </rhAlgebraicExpression>
</complexExpression>
```

Figure 4.7: Example expression using a division operator

4.1.3.3 Sets

The intended scope of a set is close to that of the noun as part of speech—a person, place, thing, action, concept, event, quality, state etc. Every model must have at least one set element. An example of a set is a collection of securities, or a collection of time periods. AML supports two types of sets: *complex* and *simple* sets. A *simple* set is best thought of as an irreducible concept i.e. a basic set which is not dependent upon other sets. A *complex* set on the other hand is one which can be dependent on or more other sets, simple or complex. Figures 4.8 and 4.9 provide examples of simple and complex set definitions respectively.

```
<set>
       <simpleSet>
              <setName>Equities</setName>
              <documentation>
                     <onTextComment>
                            This set represents the candidate equity portfolio
                     </onTextComment>
              </documentation>
       </simpleSet>
</set>
```

Figure 4.8: Simple set declaration

```
<set>
	<complexSet>
		<setName>Portfolio</setName>
		<rhs>
			<lhSet>Equities</lhSet>
			<setOperator operand="union"/>
			<rhSet>Bonds</rhSet>
		</rhs>
	</complexSet>
	<documentation>
		<onTextComment>Candidate mixed portfolio</onTextComment>
		 <offTextComment>
			Sample complex set declaration
		</offTextComment>
	</documentation>
</set>
```

Figure 4.9: Example complex set declaration

4.1.3.4 Attributes

AML supports three attribute types: *scalars*; *parameters*; and *variables*. A *scalar* represents a constant or never changing value. It is fixed at the time of model definition and can only be changed by altering the model. Figure 4.10 provides an example of a scalar definition.

```
<constant>
	<scalarName>Rf</scalarName>
	<rhs>
		<scalarValue>3.75</scalarValue>
	</rhs>
	<documentation>
		<onTextComment>Riskless rate of return</onTextComment>
	</documentation>
</constant>
```

A *parameter* represents a value-bearing property which can be defined in connection with a set or other attributes. In essence, it provides a means of assigning values to the concepts encapsulated in a set. Figure 4.11 is an example of a basic parameter declaration.

Variable is a self explanatory term, it refers to an attribute type whose value is placed under the control of a solver or a target environment. Figure 4.12 illustrates a variable declaration.

```
<parameter>
	<name>Nt</name>
	<rhs>
		<expression>
			<lhExpression>
				<elementReference>
					<attributeName>Dv</attributeName>
					<attributeIndex>
						<indexString>t</indexString>
						<setName>securities</setName>
					</attributeIndex>
				</elementReference>
			</lhExpression>
			<operator>*</operator>
			<rhExpression>
				<elementReference>
					<attributeName>Dm</attributeName>
					<attributeIndex>
						<indexString>t</indexString>
						<setName>securities</setName>
					</attributeIndex>
				</elementReference>
			</rhExpression>
		</expression>
	</rhs>
	<documentation>
		<offTextComment>Parameter declaration</offTextComment>
	</documentation>
</parameter>
```

Figure 4.11: Parameter declaration

```
<variable>
	<name>amountInvestedInSecurity</name>
	<index>
		<indexName>i</indexName>
		<setName>Portfolio</setName>
	</index>
	<lowerBound>
		<comparator>greaterThan</comparator>
		<boundValue>
			<float>0.0</float>
		</boundValue>
	</lowerBound>
	<upperBound>
		<comparator>lessThan</comparator>
		<boundValue>
			<float>0.4</float>
		</boundValue>
	</upperBound>
	<documentation>
		<onTextComment>
			Amount invested in each security from the
			candidate portfolio
		</onTextComment>
	</documentation>
</variable>
```

Figure 4.12: Variable declaration

4.1.3.5 Functions

AML provides syntax for representing both constraints and objective functions. The major difference between a constraint and an objective function is that the latter doesn't have a right hand side value i.e. doesn't make use of equality or inequality operands. Objective functions are not mandated by the AML grammar, and as such it is easy to represent a constraint programming model in AML. The following figures provide examples of constraint and objective function declarations respectively.

```
<constraint>
	<constraintIdentifier>
		<constraintName>
			Total Amount Invested In Securities
		</constraintName>
	</constraintIdentifier>
	<constraintFunction>
		<expression>
			<lhExpression>
				<elementReference>
					<attributeName>Dv</attributeName>
					<attributeIndex>
						<indexString>t</indexString>
						<setName>securities</setName>
					</attributeIndex>
				</elementReference>
			</lhExpression>
			<operator>*</operator>
			<rhExpression>
				<elementReference>
					<attributeName>
						decisionVariable
					</attributeName>
					<attributeIndex>
						<indexString>t</indexString>
						<setName>securities</setName>
					</attributeIndex>
				</elementReference>
			</rhExpression>
		</expression>
	</constraintFunction>
	<comparator>equal</comparator>
	<rhs>
		<literal>1</<literal>
	</rhs>
	<documentation>
		<onTextComment>
			total amount invested in securities
		</onTextComment>
	</documentation>
</constraint>
```

Figure 4.13: Example constraint declaration

```
<objective>
	<target>maximize</target>
	<objectiveName>Optimized Portfolio Return</objectiveName>
	<objectiveFunction>
		<expression>
			<lhExpression>
				<elementReference>
					<attributeName>
						decisionVariable1
					</attributeName>
					<attributeIndex>
						<indexString>t</indexString>
						<setName>securities</setName>
					</attributeIndex>
				</elementReference>
			</lhExpression>
			<operator>*</operator>
			<rhExpression>
				<elementReference>
					<attributeName>
						decisionVariable2
					</attributeName>
					<attributeIndex>
						<indexString>t</indexString>
						<setName>securities</setName>
					</attributeIndex>
				</elementReference>
			</rhExpression>
		</expression>
	</objectiveFunction>
</objective>
```

Figure 4.14: Objective declaration

4.2 ORML (Optimization Reporting Markup Language)

ORML is the companion technology to AML. Whereas AML is concerned with the representation of optimization models, ORML is concerned solely with representing the information that is generated as a result of manipulating those models. Unfortunately the differences in representation formats are not limited to just models or model instances, but also extend to the representation of results, where the term ‘results’ refers to information that is generated by solving or analysing a model. What ORML proposes to do, is to abstract the representation of this information. Because ORML like AML, deals with the abstraction of representation, it can be argued that both should be treated as two halves of the same technology or concept, however it is entirely likely that an organization or individual may use AML on its own, perhaps for the purposes of model sharing or storage.

In essence, AML representation need never actually be used directly in computation, but rather can be translated to other formats which suit the user’s computational environment. If both are treated as the same technology and distributed as such, cases where they are used individually are bound to raise serious questions about the validity of such a strategy. For example consider the case of a Prof. Morgan, who is an OR lecturer. For reasons of portability she always specifies her models in AML, and insists that her students do the same. The reason for this is because she and her students use different modelling environments, and as such require a format which will allow them to share models easily. So if for example she uses the ZX algebraic modelling system, she can transform models submitted by her students

to the ZX format using a ZX translation toolkit downloaded from an open source repository and then execute the model within her own modelling environment. In essence, she and her students are using AML purely for sharing models, and never actually execute the model in AML format. Hence the issue of results never really arises. In such a scenario, if ORML and AML were bundled as part of the same technology, then the professor and her students would have a lot of redundant functionality which they do not make use off, and which from their point of view would only serve to increase the size and complexity of the technology.

For this reason, ORML is published as a different and independent technology which can be used in conjunction with AML to guarantee true portability of representation, not only of models or model data but also results.

4.2.1 Background

Representational problems are not limited solely to models, or instance data, but also extend to the results of performing computations on such models or their instances. This arises due to vendor specific interface requirements. By definition, interfacing does not only involve data input, but also data output. This implies that if the input or the protocol involved in supplying it is specific to a particular piece of software, then the output from such software is bound to be specific to it as well. Invariably systems wishing to make use of such software become dependent upon it, and any representation formats which it uses, thereby complicating the task of integrating similar or related software into the solution landscape.

AML only provides half of the solution to the problem of abstracting away from vendor specific data formats. Whereas it provides a means of abstracting away from formats for input data i.e. models and model data, it doesn't address the problem of obtaining the results of computations performed on such models or their instances. Not addressing this problem would lead to a situation where vendor specific representations have to be interrogated in order to obtain the results of computations. Apart from the fact that this would almost certainly wipe out a lot of the gains made by using a format such as AML for input representation, it also means that the user's environment is once more coupled to the underlying optimization system. Such a scenario is illustrated by figure 4.15. In essence, even though a portable representation format is used by the client system to represent models, it still has to be aware of the downstream optimization engine in use and its specific format requirements, and it will have to use API or syntax specific to such a system in order to obtain the results of computation. The arguments for using a scheme such as this are likely to find little or no sympathy among developers or modellers, as there would be no benefit gained from investing in the effort to use a portable representation.

Using vendor specific result representation formats also complicates the task of delivering optimization functionality via the Internet. Apart from the fact that the vendor representation has to be encoded in a fashion which is suitable for Internet transport, application clients have to make use of API specific to the vendor in order to access this representation. The pitfalls of such a scenario are best illustrated with a hypothetical case. Consider the case of *OptiSoft Inc.*, a small specialist software house which produces the *iOpt* solver. In order to reduce its distribution costs and the risk of piracy, and to generate higher revenues through a service metering approach to licensing, its management has chosen to distribute the iOpt functionality over the Internet as a web service. So as to gain a foothold in the market, and to avoid being edged out by their larger competitors, they decided to adopt AML as the de-facto format for representing input to their iOpt web service.

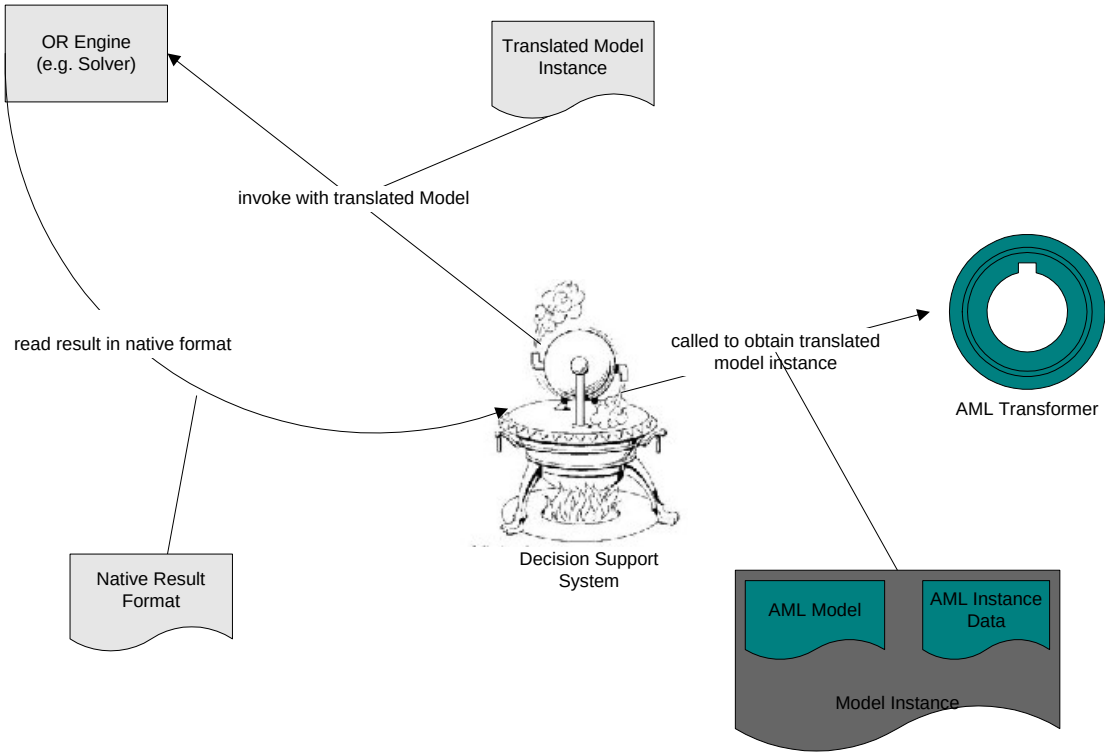

Figure 4.15: Direct result access

However as there is no portable means of representing solver output, they resort to using an encoded version of the native iOpt output. This results in a situation where OptiSoft customers implement software specifically for manipulating the iOpt output format. Inundated with support calls related to their result representation format, and requests for more education and training material, the board decide that it would be in the strategic interests of the company to implement a parser library to help their customers integrate the iOpt service easily into their infrastructure. This of course means that OptiSoft will once more be in the business of distributing iOpt related software through the conventional mechanisms as opposed to doing so via web services. Issues which it had to sought to avoid begin to re-appear again, such as managing release cycles, interface changes, etc. It also means that OptiSoft customers are once again tied to a single vendor and the vendor's representation format. As such, integrating an additional solver from a different vendor will ultimately involve time and money, not to mention the risk that is introduced by switching to a different result manipulation API. It may even be possible that the new solver does not provide information which is similar to that of iOpt, and as such the customer(s) in question may have to re-define the way in which the results are presented.

Examining the scenario depicted by figure 4.15 and considering the above example, it is clear that a non portable format for result representation, leads to a scenario where direct dependencies are established between the application client and third party software. In the example above, even though OptiSoft and its customers have chosen a portable format for representing models and model data, the problems which they sought to avoid reappear because they are using a non-portable format for representing results.

ORML provides a solution to this problem by abstracting away from formats and syntax utilised by specific software and/or vendors for representing result data. Like AML, it is XML based, and essentially defines a syntax or language for representing result data, independent of the source. As such, it provides applications with a single integrated view of computation results. Computation results not only refer to solution data, but also infeasibility information

and analysis results. Analysis in this sense not only refers to post-optimality analysis information, but also to pre-optimality analysis including model validation.

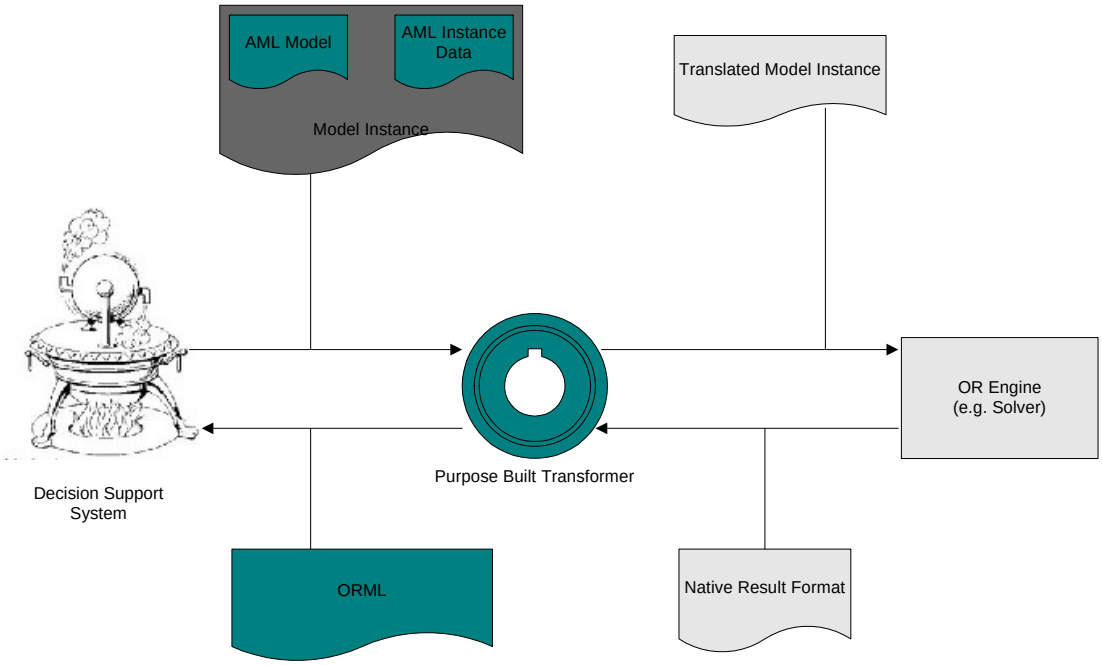

Figure 4.16: Optimization engine access via a translation toolkit

Combining AML and ORML leads to an architecture where the syntax specific to the underlying optimization framework is completely invisible to the client application. Input can be represented in AML format, and using a transformation toolkit transformed into a format expected by the underlying OR engine, the output from this engine can be transformed by the same toolkit into ORML format. Therefore the application's view is limited to the translation toolkit, thus shielding the upstream application components from changes to the underlying OR engine. This scheme is illustrated by figure 4.16. This is particularly true for services delivered over the Internet, as there will be no requirement to provide a transformation toolkit for the vendor software, because access is via Internet protocols. In essence if the service interface is defined in terms of AML and ORML, application developers have to do little or no work to make use of or integrate the service. In the case of software that is installed and accessed locally i.e. resides within the same infrastructure as the application client, the application client's view can be limited by a proxy which encapsulates the specific transformation toolkit required for the software. This will further insulate the client from changes to the toolkit. Even in cases where this is not done, the application client's view of results is not affected by changes to the downstream OR software provided a transformer exists for the target software, or the target software provides direct support for ORML.

In summary, the ORML format provides applications with as single unified view of result information, irrespective of the source of the result. It is based upon XML and as such supports the generation of multiple views from the same content. Unlike AML, ORML content will typically either be created directly by OR software or a transformation library will have to be used to convert software specific data into ORML format. This complicates the structure of ORML slightly as different software use different schemes for representing result information. Furthermore there is no uniformity of the data set that is returned by various software. Therefore ORML has to be robust and generic enough to cater for differences in representation.

4.2.2 Advantages of ORML

The advantages of the ORML approach to result representation can be summarised as follows:

i. ***Unified view of results:*** It provides applications, application clients and to a wider extent users with a single unified view of computation results irrespective of the source. Developers and modellers no longer have to provide specialised code to deal with the various representation formats employed by different software. Rather, it is possible to utilise a common code-set for manipulating results. More importantly, this would lead to a situation where changes to the underlying third party OR software have little or no impact on upstream application components. The interfaces of such components are limited to the ORML representation of results, which in turn can be built up from native formats. Consequently the application and its constituent components are completely de-coupled from the result representation format employed by downstream OR software.

ii. ***Generality:*** It provides support for the known classes of optimization and constraint programming problems. It is not limited to a particular class of models or a specific paradigm. It is also robust and versatile enough to encapsulate the result representation format employed by a variety of optimization software in today's environment, and aims to cater for future formats.

iii. ***Support for multiple views and report generation:*** Judging by its name, ORML is clearly a language designed to support reporting. Due to the fact that it is based on XML it is possible to generate a variety of views of result information either by using relatively simple stylesheets or specialised transformation engines. To illustrate, consider the case where the result data is being used in a decision making environment, it is very likely that technical staff such as OR practitioners would prefer to view the technical aspects of the result, the managers a detailed management style report, and other people involved in the process may just want a summary or an overview. Three different stylesheets can be applied to the same result data to generate the views required by these three different groups of individuals. In an environment where OR plays a large part, it is quite likely that these stylesheets would be pre-defined beforehand and as such can simply be applied to the results as soon as they are obtained in order to quickly generate the reports/information required by the different participants in the process. This not only saves time, but offers practitioners the flexibility to utilise results for a variety of purposes and in a variety of ways. It also completely eliminates the need to re-process models in order to obtain different views of what is essentially the same information, or the need to build specialised software to generate different views of result information.

iv. ***Efficiency:*** With some traditional modelling tools or OR software it is necessary to repeatedly process models in order to access different sets of result data or to present data in different formats. ORML eliminates the need to do this as it makes it possible to represent the entire result data set in a densely populated XML structure which can then be repeatedly queried or transformed into alternative views.

v. ***Portability:*** Although portability is not a key requirement of result representation, it is nonetheless an important advantage offered by ORML. It is quite likely that results will be used in different environments including operating systems, perhaps stored in different locations such as relational databases. Therefore it is important that the result format is portable so that it can be utilised or stored on any environment without loss of information. Consider the case where a model solution forms part of the input to a critical corporate decision, it is very likely that the solution will be saved in a location, possible written to a database, and referred to in subsequent management meetings and presentations. In such a scenario the advantage of having a portable presentation becomes clear.

4.3 WSOP (Web Services Optimization Protocol)

WSOP is a recommendation which aims to specify a common lingua-franca for conducting operations research transactions over the Internet. It builds upon the typing mechanisms provided by ORML, and AML to define a set of data exchange guidelines and interfaces which allow the seamless discovery and integration of operations research functionality. One of the goals of distributed computing and in particular web services is to enable a situation where disparate computer systems can communicate effortlessly without the need for code changes or vendor specific interface implementations. WSOP aims to achieve this goal for operations research software by providing a common means of describing, invoking and managing functionality. It also aims to simplify on the part of the provider, the process of describing and advertising such functionality.

It achieves its aims by the identification and cataloguing of operations research processes and the partitioning of the operations involved in such processes into logical groups according to the roles which they play in the process. It forms the last part of the OOF trilogy/set that can be used to enable the seamless access and distribution of optimization functionality over the Internet. The other two are AML, and ORML which provide the typing mechanisms utilised by WSOP. Combining the three enables organisations to publish OR technology over the Internet, and customers to easily access such technology. This is because AML and ORML both provide independent and portable representation formats for representing data and when combined with the generic set of Interfaces defined by WSOP leads to a situation where enterprises always have the same view of services irrespective of the provider or the underlying technology employed by the provider.

This accomplishes the single most important objective of the WSOP initiative, which is to enable a situation where application clients can automatically discover and invoke optimization services via common Internet protocols without the need for a specific integration exercise or any form of human intervention. WSOP in achieving this also simplifies the process of describing, advertising and managing optimization services. Consequently preventing the domination of the market by a handful of large players and the proliferation of vendor specific approaches—interfaces, schemes etc. for publishing functionality on the Internet. It is neither a standard nor is it yet a proposed standard, but rather a recommendation on how best to solve the problems encountered in delivering or accessing operations research functionality over the Internet, in order to exploit the opportunities which the Internet and distributed computing technology provides.

4.3.1 Background

The goal of delivering operations research functionality over the Internet is obscured by a number of obstacles, some of which are symmetrical to problems which users or organizations would face in attempting to access such functionality. These can be grouped under broad headings such as: usability, flexibility, data definition, security, management of services etc. These problems are not strictly limited to operations research alone nor do they originate from it. In fact, the problems associated with conducting business over the Internet have led to initiatives such as ebXML x[11] and OASIS [36] which aim to facilitate global trade by delivering a common set of internationally agreed upon XML semantics and related document structures, that govern business data exchanges over the Internet. However the objective of delivering operations research functionality over the Internet has only recently materialised, and as such little discussion has gone on as to how best to achieve this. Issues with which the wider computing and Internet commerce computing community have been grappling with for a relatively long time have as yet not been considered by the OR community. Delegating the task of resolving these issues to vendors is likely to result in a situation not that much better than the current one, where a large number of vendor specific schemes exist for solving what are essentially common and reasonably straightforward

problems. This not only results in duplication of effort and the wasting of resources, but ultimately complicates the task of purchasing or utilising OR functionality delivered via the Internet. It could also lead to a situation where a few large players dominate the marketplace making it difficult for newer or smaller players to gain a foothold.

The WSOP initiative aims to stimulate discussion of these issues and more importantly at recommending a collective solution to them. Despite its name, WSOP is more than just a message exchange protocol; it seeks to provide architectural solutions to problems which are involved in distributing operations research functionality over the Internet. It is not intended as a standard, but rather a recommendation of how to solve common problems. It is structured in such a way so as to enable it to be used in application-to-application communication within an enterprise, as well as in the larger Internet arena. It is based on open international standards such as XML, WSDL and SOAP, and utilises the typing technology offered by AML and ORML.

4.3.2 WSOP Motivation

In order to fully appreciate WSOP, it is important to explain some of the issues which motivate it. It is not possible to cover the full range of issues which the recommendation tackles, as they do not fall within the scope of this paper. However a summary of the key issues is provided in this section.

4.3.2.1 Usability and Flexibility

These are perhaps the most difficult problems to overcome in order to effectively deliver operations research functionality via the Internet. Publishing software over the Internet in itself is not a difficult task given the current array of tools in the software market which are aimed at doing just this. The problem though is in what state the software is published, and how usable it is. If anything, an interface which is very complicated, fault intolerant or error prone will discourage users from making use of it. Therefore the real challenge is to ensure that services are described in such a way that they are intuitive, easy to use and robust. Given enough demand, and adequate resources, vendors are bound to achieve this, if for no other reason but to encourage the purchase of their services. However, in the case where vendors provide their own specific solutions i.e. define interfaces and architectures specific to the technology which they are offering, a new and perhaps even more damaging problem would arise—*inflexibility*. The best analogy for this problem is that of the language barrier faced by tourists. Whereas an interface provided by a particular vendor might suit its purposes, if a user wishes to switch to another vendor, the user would have to re-write parts of their system, in just the same way a tourist generally has to learn the basic words of the language spoken in whichever country they are visiting (or at is advised to do so). It also becomes more difficult to implement solutions which require the integration of two or more services. This is because these services will ultimately have their own specific interfaces, possibly with their own typing conventions. In such a scenario, software bridges would have to be put in place to enable the combination of such services.

WSOP seeks to solve the problems of usability and flexibility by first identifying and cataloguing the processes which providers and users are likely to be involved in and extracting the information needs of such processes, and finally defining the interfaces required for them based on the data requirements, and the data flow between the parties involved in the process. For example, take a very simple process such as invoking a solver over the Internet. Regardless of the software on either side of the divide i.e. client or server, a representation of the model and its relevant data have to be supplied to the server, and a representation of the results have to be returned by the solver. Given this requirement for three data items, and the flow of these items, it is plain to see that an interface which accepts a model and its data as input, and returns a solution structure is required. This is a very simple example but serves to illustrate the point. First the process is identified i.e. obtaining a model

solution. Then the information required for the process i.e. model, model data, and a solution are extracted from the process. These data and their associated flows are mapped to an interface which accepts a model and model data as input and returns a solution object. The means of doing this is borrowed from software engineering and as such should be fairly intuitive to software engineers.

As already mentioned, it is quite feasible for individual firms to use similar approaches to define their own interfaces; however the end result of this is that the interface set will once more end up fragmented. WSOP attempts to avoid this situation by defining a set of core but ultimately extensible interfaces. By making the interfaces extensible, vendors can add extensions that tie in more closely with their application model.

4.3.2.2 Data Definition

Within a given application context, an interface is unique due to its identifier or name, the input(s) it receives and the output(s) it returns. The combination of these three items is often referred to as an interface *'signature'*. Therefore the first step in actually homogenising interfaces to operations research software is to determine a representational format for the data i.e. a typing mechanism. Even if there was no need to achieve flexibility or homogeneity, there would still be a need for vendors to utilise typing mechanisms for transmitting information over the Internet. The temptation is for every vendor to define its own types and data structures and of course its own interfaces. There might even be a temptation among older and more established vendors to 'shoe horn' their existing data structures into a form that is more suitable to Internet messaging. Either of these solutions is bound to lead to more confusion than exists at the moment. Application developers and users would once more be forced into using vendor specific schemes, with all the attendant drawbacks of such arrangements. In addition to defining a format for representing the data, there also needs to be some definition or description of what the data means i.e. the real world concept that it embodies or encapsulates.

WSOP provides a solution to the problem of data representation by building on the flexible, extensible, and portable type syntax provided by ORML and AML. WSOP in itself also provides type definitions for control data, i.e. data which is not directly related to the concepts of modelling embodied by AML and ORML but which are required for the successful processing of transactions and the management of services.

4.3.2.3 Security

Security is a problem which has haunted the Internet community almost as long as Internet based communication has existed. In fact the security of electronic communication pre-dates the Internet era and is likely to remain an issue for the considerable future. As such it is a recurring theme in material covering distributed systems, especially those which make significant use of the Internet or Internet-based protocols. It is a multi-faceted topic of which some aspects are not directly applicable to web based decision support services. This paper will only cover authentication which in the view of the authors forms one of the fundamental security requirements of an Internet based model for distributing software.

Authentication involves verifying the authenticity of user identification. It also involves access control which in turn is concerned with maintaining and enforcing user privileges and rights. There are a variety of authentication schemes, each with different architectural, infrastructure and implementation requirements. The goal of WSOP is to specify the most efficient and cost effective authentication mechanism which is applicable to the OR software delivery model. In order to achieve this, it is important to identify the areas where authentication is required and what sort of authentication is required.

4.3.2.4 Advertising

Service providers ultimately need a means of advertising their software or services to potential customers. Operations research is a specialist market and as such some services do not have enough mass to justify the high cost of traditional advertising media, or are simply unsuitable for such media. WSOP provides a new means of advertising software by recommending a structure for UDDI registries of optimization software. This not only enables service providers to advertise at a much reduced cost, and at a targeted audience, but also symmetrically allows users to easily search for services. The structure of the registry also facilitates a situation where searches can be automated by software, thereby enabling the automatic discovery of services. Generalizing the service interfaces ensures that services are described using the same syntax thereby making it easier for providers to write to the registry, and also enables the automatic integration of services at runtime.

4.3.2.5 Service Management

The WSOP recommendation includes information on the organisation of services so as to enable easy management. Management can mean a situation where services are grouped together and managed in tandem by a management server; it could also mean the maintenance of WSOP style registries of services. It could also include tasks which a client has to perform to enable the easy use of an unmanaged service. The WSOP recommendation attempts to ensure that the same interfaces can be used to accomplish these tasks, thereby providing flexibility on the choice of deployment topology and method of access. For example a registry will need to keep its information up to date, therefore it has to occasionally check the status of services and perform cleanup operations when it is satisfied that a service or a group of services is no longer active. In the same guise, a client program which is communicating directly with an unmanaged node may want to verify the status of the node prior to sending a request to it. In the case of a service deployed within an enterprise environment, a special management service could be used to perform checks on the status of nodes.

4.4 OSCP (Optimization Service Connectivity Protocol)

The OSCP recommendation is aimed initially at providing a Java abstraction of optimization software interfaces. There is a variety of software utilised in a decision support process and each of these dictates its own specific interfacing strategy. Majority of the interfacing requirements are vendor, or even sometimes product specific. This complicates the task of utilising products from more than one vendor or utilising more than one product. Integration of different products, possibly from different vendors is often a costly and tedious exercise.

OSCP provides a standard means of invoking optimization software independent of the specific interfacing requirements of such software. This is achieved by providing a high-level abstraction of the software interface and delegating the implementation of low level details to another piece of software called a *'driver'*. As opposed to the current scenario where application developers have to implement software bridges for each system that makes use of optimization technology, these drivers can be implemented once and re-used in multiple scenarios. To fully understand the impact of this, consider a case where a decision support system or even a modelling system has to interface with N different products. This requires the implementation of N specific interface bridges. If M of such systems is implemented, then M x N interface bridges are required. The OSCP recommendation aims to bridge the interfaces of client applications with that of operations research software. This is achieved by limiting the view of software to the OSCP implementation. In essence client applications no longer interact with operations research software directly but with the OSCP implementation. This is then responsible for invoking the driver for the target software, where the driver is

responsible for low level interfacing. Therefore there need only be one generic driver for each software product, and this can be re-used in a multiplicity of scenarios. As opposed to the current scheme where M x N interface implementations are required (excluding platform dependencies), the OSCP recommendation reduces this number to N. In essence N drivers for N product interfaces. The implementation of the driver can be undertaken by the product provider, another party, or possibly even an open source initiative.

OSCP in essence abstracts away from product interfaces, and recommend a vendor neutral approach to integrating software. It achieves this by relying on the typing mechanisms provided by XML, AML and ORML. These make it possible to abstract away from vendor specific data representation schemes.

OSCP is implemented in Java to guarantee portability across different computing platforms. Although it is possible to implement the recommendation in other languages, Java was selected as the best option as it fits into the vendor independent approach espoused by the Open Optimization Framework.

4.4.1 Background

Real world applications of operations research theory usually require the combination of different pieces of technology, possibly from different vendors. A decision support system for example is likely to have one or more embedded solvers, one or more pre-solve routines, and possibly one or more analysis tools. Each of these has different interfacing requirements i.e. each specifies its own interface, which are often vendor specific. ‘Interfaces’ in this sense also refers to the data structures used by the software to represent information.

Interfacing to a piece of software involves implementing the calls to the software, and converting internal data representation structures into those expected by the software. This is bearable where the number of such software is kept to a minimum, preferably one. However, when this has to be repeated for a variety of software, or where the software in question is likely to change, then the problem becomes unbearably expensive and time consuming to solve. In the case of decision support systems or systems that are dependent on operations research software, the problem is further complicated by a variety of factors. The software used is often dictated by the problem being solved. The software and the problem both dictate the internal data structures utilised by the system. To illustrate with an example, consider a portfolio optimization system which has as its backend an optimization solver. The system makes use of a linear model, which dictates that a linear solver must be used. The linear solver specifies a representation format for model instances, and as such the system uses this representational format. In the case where the system’s internal model changes to a non-linear one, the solver will have to be changed. The new solver will dictate a new representational format, which the system has to be ported to.

The example above is a relatively simple and straightforward one, consider the case where the model in question is loaded and executed at runtime. This implies that a stable of solvers have to be maintained to cater for all model possibilities. Each of these specifies a different interface and different data representation structures. Consider the case where an analysis tool is thrown into the mix. This may have its own solvers, or may even be dependent on a modelling system. It is more than likely to specify its own interface and its own data structures. Integrating all these systems together posses a very difficult challenge. Each has its own interface requirements and data representation format.

The solution is to abstract away from the interfaces and data representation formats used by all these systems. In essence provide application clients with a uniform view of operations research software, irrespective of the specific interface or data representation requirements of such software. A not too dissimilar problem was faced by early application developers who

had to interface to relational database systems. Each database provided its own set of interface libraries, thereby complicating the task of using multiple databases or ensuring the portability of the code from one database system to another. The problem was even more profound in the early days of database programming before the emergence of SQL, where each database provided its own syntax as well. This problem led to the birth of interfacing technologies such as ODBC and JDBC. These provided application developers with a single unified view of database systems, irrespective of the specific interfaces supported by such systems. The only requirement being that the target databases support SQL. The details of low level interfacing are delegated to another library called a 'driver' which today is normally supplied by the database vendor. The driver however can be implemented by any other party, and indeed there are a number of drivers which have been produced by open source initiatives. The use of ODBC and JDBC in essence provided developers the ability to integrate different databases at runtime without the need to make any code changes. Drivers and connection details could be specified at runtime, and these would be used by either ODBC or JDBC to connect to the target database. This not only simplified integration, but also guaranteed the portability of the application from one database system to another.

Applying a similar concept to operations research, it is possible to envisage a scenario where interface abstractions for software such as solvers and analysis tools are used as a proxy to such software. The major obstacle to this is the fact that the various software make use of different data representational formats. In the case where these representational formats can be abstracted, then the problem becomes relatively straightforward to solve. AML and ORML both provide abstractions for representation formats, thereby making the task of interface abstraction easier. OSCP exploits the representation mechanism provided by ORML and AML, to create an abstraction of operations research software interfaces. It is implemented in the Java language and as such can be used in a variety of platforms/environments.

The OSCP recommendation makes use of the delineation of roles and responsibilities approach dictated by the framework. As such, the organisation of software interfaces is identical to that used by WSOP. In essence an interface is a logical concept which maps onto the role performed by a piece of software. So if for example the software is responsible for obtaining a solution then it will have a solver interface. If on the other hand it is responsible for pre-solve analysis, then it will map onto a pre-solve analysis interface. There is no limit on the number of roles performed by a piece of software; however drivers have to be provided for the various interfaces which these roles map onto.

The task of low-level communication with the target software is in essence delegated to the driver. The client application communicates strictly with the OSCP implementation, although facilities are provided to use vendor extensions if application developers wish to do so. This means that new software can be integrated by simply specifying a new driver; thereby eliminating the need for code changes. Figure 4.17 provides an illustration of the OSCP architecture. In summary a *'DriverManager'* is responsible for maintaining a list of drivers, and this allows the dynamic registration and retrieval of drivers. A driver abstraction is provided for the various interface types such as solvers and validators. The diagram shows that there can be a number of driver types each mapping onto a role as defined by the framework. For example there can be a *'SolverDriver'* and a *'ValidatorDriver'*, where the latter is responsible for interfacing to a component which performs model validation. A driver returns a *'ServiceConnection'* to the target software. In the case of a solver, a *'SolverConnection'* is returned, and in the case of a validator, *'ValidatorConnection'* is returned. The connection represents an abstraction of the target software, and all communication with the software is performed using the connection object. The connection and driver interfaces in essence represent abstract concepts that expose methods based on AML, and ORML. Driver implementations provide concrete representations of these interfaces which are specific to the target system.

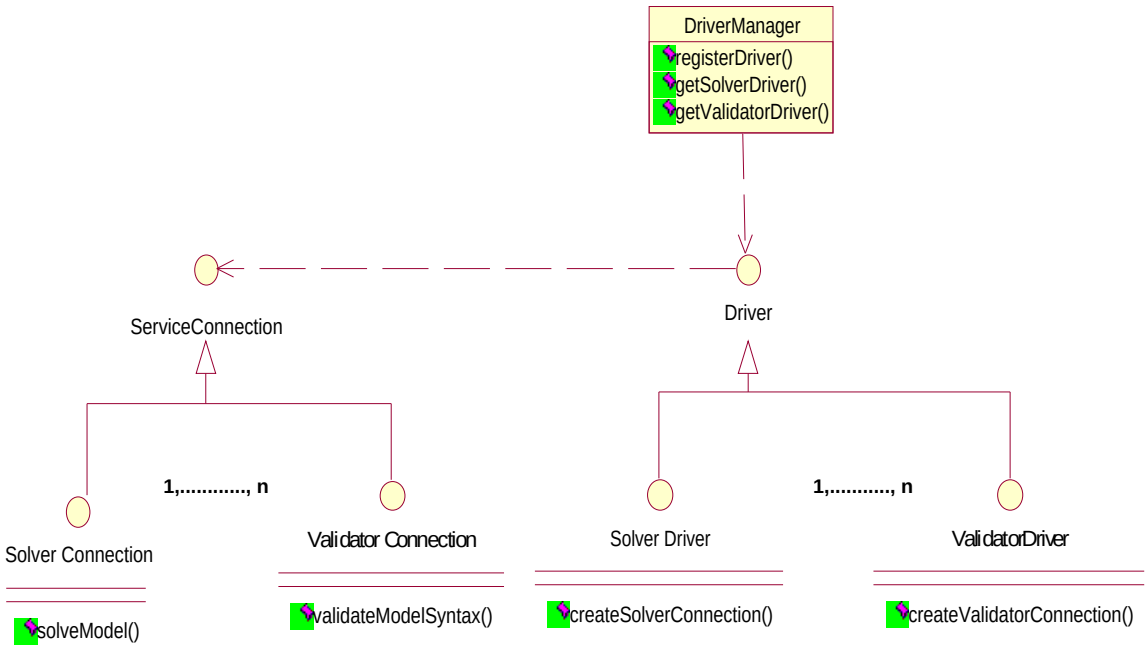

Figure 4.17: Illustration of the OSCP architecture.

OSCP is not a standard, but rather a recommendation on how to ensure the interoperability of operations research software. The motivations behind the recommendation are summarised in the following section.

4.4.2 OSCP Motivation

The OSCP recommendation is still at an early stage in its lifecycle. It is quite likely that the requirements addressed by the recommendation will alter from time to time before the final draft due to contributions from external parties, and due to issues raised during the implementation process. However the key requirements or drivers behind the recommendation will remain the same. These are summarised below.

4.4.2.1 Ease of integration

The major motivation behind the OSCP recommendation is to ease the integration of operations research software. It is very likely that a variety of software will be combined to solve real world problems of any significance. It is also likely that software will have to be integrated into the existing IT infrastructure of the organisation wishing to implement a solution. This more often than not is an expensive and time consuming exercise. The OSCP recommendation provides an easy and well structured method of integrating software by abstracting away from the interfacing requirements of each individual product.

4.4.2.2 Interoperability

Different software specify different interfaces and data structures, thereby complicating the task of utilising multiple software in tandem. The requirement for combining software is one that often arises in real world applications of operations research theory. The problems associated with meeting this requirement become clearer if we consider a situation where the output from one software is required as input to another. If both use different data structures or data representation technology, combining both would require a programming exercise to convert the data structures utilised by one into that expected by the other. This is simple enough for two static pieces of software, however consider the case where the combination of software is dynamic or has to be performed at runtime, perhaps directed by user input. Consider for example a supply management system, which can integrate into one or more graphical tools used for presentation. The choice of the presentation format is dictated by the

user based on individual cognitive preferences therefore the output from the internal solvers would have to be converted into a format expected by the chosen graphical tool. The solution to this problem would involve building several components, each responsible for generating the data input expected by each possible presentation tool. This example gets more complicated when there is more than one solver involved, each with a different output format.

The OSCP recommendation provides an easy means of providing interoperable solutions. A single and straightforward typing mechanism is provided by using AML and ORML; consequently client applications can rely on a single data representation format. Furthermore, the interface to each target software is abstracted, and the client applications have a single unified view of all software irrespective of the specific interface requirements of each.

4.4.2.3 Portability

In order to achieve its aims the OSCP recommendation must not have any vendor dependencies. This not only goes against the principles of interoperability and ease of integration, but inadvertently introduces clashes of interest. Instead the recommendation should be free of vendor specific constructs, schemes or concepts, and should aim for a scenario where the success of the framework is not dependent on vendor support.

Also the recommendation should not be dependent on any particular platform or operating system. For this reason it is implemented in the Java programming language which should enable portability across a variety of platforms. The typing mechanisms employed by it are XML based, and as such should guarantee the portability of data.

5 Future Direction

The output of research projects most likely have to make a transition to the non-academic world at one stage or another if they are to be employed in solving real world problems. It is quite likely that the output of a project may have to be revised or tailored in order to meet more closely the requirements of the commercial world. In some cases the revision process is an ongoing exercise, in which case a means is required to dictate the interval at which such revisions are made.

The framework in its current state includes two recommendations which are yet to be finalised. In order to finalise these, the participation of other members of the academic arena is required, and also those of the commercial world. The process of modifying the recommendations or indeed the entire framework has to be performed in a controlled manner, and as such there needs to be a management process for controlling contributions and implementing suggestions. Therefore the two major tasks that have to be undertaken in the near future is to set up a development forum and establish the guidelines for participating in this group, and secondly set up a management process to control the evolution of the framework.

It is difficult to overstate the case for participation given the nature of the problems being tackled. Input is required from a wide variety of professionals including members of the academic and commercial world. This is particularly because the framework should be organised in such a way that it is easy to use in practical scenarios, and is robust enough to meet the requirements of today’s computing environment. Ideally participation should not be limited to either representatives of the commercial world or members of the academic community. This is to ensure that the best and brightest individuals are free to contribute to the further development of the framework irrespective of affiliation.

Obviously any gathering or organisation which does not possess any sign of leadership is very prone to dysfunction. Therefore there is a clear and visible need to establish some sort of authority to govern the participation process. This authority will most likely be in the form of a working group or council. Its responsibilities will include: defining a process by which individuals can contribute to the project; managing the implementation or inclusion of ideas, changes to the framework; and more importantly managing the process by which new versions of the framework are published to the external world i.e. the versioning or release process. It is envisaged that once the working group is established the framework will become more or less an independent entity with a well managed evolution process.

6 Conclusion

The major motivation behind the numerous efforts to establish a new representation format for optimization models and model instances is the limitations associated with current formats. In addition to solving these problems, it is expected that any new representation format will exploit advances in modern computing technology, particularly in the areas of Internet and distributed computing. This paper has identified the problems associated with model representation schemes, and outlined the major opportunities which the Internet provides to operations research. It has also examined current initiatives aimed at establishing new representation formats for optimization model instances, and outlined the weaknesses of these initiatives. Whereas they are suitable for representing instances of specific classes of optimization model, they do not posses the features for a general representation format capable of covering the known classes of optimization and constraint programming models, and their associated instances. They are also not suitable for decision support environments, nor do they fit closely with Internet based optimization. This is particularly due to the fact that they are tied to specific paradigms such as linear or mixed-integer programs, and they do not provide for the easy re-use of models. In summary these initiatives are aimed and are highly suitable for representing instances of specific types of optimization models.

The framework described in this paper not only possesses features to enable the successful representation of the majority of known optimization and constraint programming model classes, but also the data required to instantiate such models. It abstracts away from specific representation formats and as such offers a portable means or representing models, model instances and solution or result information. In addition to providing portable representation formats, this framework also includes a number of recommendations based on standards and technologies such as WSDL, SOAP, UDDI and Java which aim to simplify: distributed optimization in general and in particular Internet based delivery of functionality; and the task of utilising optimization software in real world applications.

The framework therefore provides an integrated solution to the problems of model and instance representation, Internet based optimization, and the use of optimization software, thus clearly meeting the requirements of a new representational format for optimization and indeed constraint programming models.

7 Bibliography

1. F. Berman, *"Grid Computing - Making the Global Infrastructure a Reality"*, John Wiley and Sons Ltd, 2003
2. George Coulouris, Jean Dollimore, Tim Kindberg, *"Distributed Systems - Concepts and Design, 3rd Edition"*, Addison Wesley, 2000
3. Hemant K. Bhargava, *"Decision Support Systems and the World Wide Web"*, http://www.smeal.psu.edu/~bhargava/, 2000

4. H. K. Bhargava, A.S. King, and D.S. McQuay, *"DecisionNet: Modeling and Decision Support Over the World Wide Web"*, Proceedings of the Third ISDSS Conference, pp. 499-506, Hong Kong, June 22-23, 1995.
5. H. K. Bhargava and R. Krishnan, *"The World Wide Web: Opportunities for Operations Research and Management Science"*, INFORMS Journal on Computing, 10:4, pp. 359-383, 1998
6. John W. Chinneck, *"Analyzing Mathematical Programs using MProbe"*, Annals of Operations Research, vol. 104, pp. 33-48, 2001
7. J.W. Chinneck. *"MINOS(IIS): Infeasibility Analysis Using MINOS"*, Computers & Operations Research, 21(1):1-9, 1994
8. *CORBA*, http://www.omg.org/corba/
9. C.J. Date. *"An Introduction to Database Systems"*, Addison Wesley, 1999
10. *DecisionNet*, http://www.ini.cmu.edu/emarket/
11. *ebXML*, http://www.ebxml.org
12. Obi C. Ezechukwu and Istvan Maros. *"AML: Algebraic Markup Language"*, (forthcoming)
13. Obi C. Ezechukwu and Istvan Maros. *"ORML: Optimization Reporting Markup Language"*, (forthcoming)
14. Obi C. Ezechukwu and Istvan Maros. *"OSCP: Optimization Service Connectivity Protocol"*, (forthcoming)
15. Obi C. Ezechukwu and Istvan Maros. *"WSOP: Web Services Optimization Protocol"*, (forthcoming)
16. Robert Fourer, David M. Gay, and Brian W. Kernighan. *"AMPL: A Modeling Language for Mathematical Programming"*, Duxbury Press/Brooks/Cole Publishing Company, 1993
17. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *"Design Patterns: Elements of Reusable Object-Oriented Software"*, Addison Wesley, 1995
18. A.M. Geoffrion. *"An introduction to structured modelling"*, Management Science, 33:547-588, 1987
19. Harvey J. Greenberg. *"A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE"*, Kluwer Academic Press, Boston, MA, 1993
20. Harvey J. Greenberg. *"The ANALYZE rulebase for supporting LP analysis"*, Annals of Operations Research, 65:91-126, 1996
21. Harvey J. Greenberg and Frederic H. Murphy, *"Views of mathematical programming models and their instances"*, Decision Support Systems, 13:3-34, 1995
22. Bjarni V. Halldorsson, Erlendur S. Thorsteinsson, Bjarni Kristjansson. *A "Modelling Interface to Non-Linear Programming Solvers. An Instance: xMPS, the extended MPS format"*, http://www.maximal-usa.com/papers/xmps/xmps.pdf, 2000
23. Bjarni Kristjansson. *"Optimization Modeling in Distributed Applications: How New Technologies such as XML and SOAP allow OR to provide Web-based Services"*, http://www.maximal-usa.com/slides/Svna01Max/index.htm
24. *HTML*, http://www.w3.org/MarkUp/
25. *The IBM Optimization Subroutine Library (OSL)*, http://www.research.ibm.com/osl/
26. *JDBC*, http://java.sun.com/products/jdbc/
27. Leo Lopes and Bob Fourer, *"An XML-based Format for Communicating Optimization Problems"*, http://senna.iems.northwestern.edu/xml/presentations/LopesFourerMiamiBeach01
28. Irvin Lustig. *"Embedding CPLEX using the ILOG CPLEX Callable Library"*, http://www.ilog.com/products/optimization/CallableLibrary.pdf
29. *Mathematical Markup Language (MathML™) 1.01 Specification*, http://www.w3.org/TR/REC-MathML/
30. Istvan Maros and Mohammad Haroon Khaliq. *"Advances in Design and Implementation of Optimization Software"*, European Journal of Operational Research 140:322-337, 2002
31. Brett McLaughlin, *"Java and XML"*, O'Reilly & Associates, Inc., 2000

32. *MPL Modeling System*. http://www.maximal-usa.com/mpl/mplbroc.html
33. *MPS Input Format*, http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/continuous/constrained/linearprog/mps.html
34. Eric Newcomer, *"Understanding Web Services: XML, WSDL, SOAP, and UDDI"*, Addison Wesley, 2002
35. *NEOS Server*, http://www-neos.mcs.anl.gov/neos/
36. *OASIS*, http://www.oasis-open.org/home/index.php
37. *ODBC*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/odp1.asp
38. *Open Grid Services Architecture*, http://www.ggf.org/meetings/ggf7/drafts/draft-ggf-ogsa-platform-2.pdf
39. *OR-Objects*, http://opsresearch.com/cgi-bin/mainIndex.cgi
40. *Optimisation Service Provision*, http://www.osp-craft.com/
41. *Oracle XML Developer's Kit for Java*, http://technet.oracle.com/tech/xml/xdk_java/content.html
42. *The SIF Reference Document*, http://www.numerical.rl.ac.uk/lancelot/sif/sifhtml.html
43. *Simple Object Access Protocol (SOAP)*, http://www.w3.org/TR/SOAP/
44. *Standard Generalized Markup Language(SGML)*, http://www.w3.org/MarkUp/SGML/
45. *The GAMS System*. http://www.gams.com/docs/intro.htm
46. *"Web Services Description Language (WSDL) 1.1"*, http://www.w3.org/TR/wsdl
47. *UDDI*, http://www.uddi.org/
48. *Xalan*, http://xml.apache.org/
49. *Xerces*, http://xml.apache.org/
50. *XML*, http://www.w3.org/XML/
51. *XML Schema*, http://www.w3.org/XML/Schema
52. *xMPS - The Extended MPS Format*, http://www.maximal-usa.com/papers/xmps/
53. *XSL*, http://www.w3.org/Style/XSL/