Numerical Optimisation as Grid Services for Engineering Design

Gang Xue, Wenbin Song, Simon J Cox, and Andy Keane
School of Engineering Sciences, University of Southampton
Highfield, Southampton, U.K. SO17 1BJ
Email: {gx, w.song, sjc, ajk}@soton.ac.uk

Abstract. In this paper we discuss the use of Grid services, an emerging Internet-based technology, to enable the application of numerical optimisation algorithms in heterogeneous, distributed systems for engineering design optimisation tasks. By being presented as Grid services, numerical optimisation algorithms can be consumed with a number of message interactions. The services are built using a combination of standard Web services and newly developed Grid technologies, based on the concept of Reverse Communication. The proposed approach eases the burden of integration by encapsulating optimisation algorithms into generic interfaces, which can be integrated into different client environments.

The design of the optimisation grid services is explained in detail, and is illustrated with concrete implementations. We also demonstrate the use of the optimisation services with real engineering design optimisation problems performed in scripting problem solving environment.

Keywords: Grid services, PSE, optimisation, Reverse-Communication

1. Introduction

Engineering design is an iterative, multidisciplinary process that is often data intensive and computationally expensive due to the application of high fidelity analysis models for the simulations of physical phenomena. In the past few decades, engineering design and problem solving have become increasingly dependent on computing, which spans from computer aided design (CAD) to simulation and visualisation. In particular, the application of numerical optimisation [1] techniques in engineering design that exploits sophisticated modelling and analysis capabilities can provide an effective measure to produce better designs in a reduced design cycle.

The increasing complexity of the simulation tools and data models involved in the design process, together with the uniqueness of each design problem that requires individual solution strategies, have made it often a daunting task for design engineers to apply specialised optimisation strategies in their daily design activities. Solving an optimisation problem usually requires the integration of various elements into an often heterogeneous and distributed design environment. It is also nec-

essary to be able to formulate different solution strategies for different design problems. In addition, easy to use is one of the most important requirements for such systems. These requirements prompted the research and development of dedicated systems for design optimisation, such as SPINEware [2], iSIGHT [3], ModelCenter [4] and more recently, FIPER [6], which attempt to provide an integrated environment for engineering design optimisation.

A common requirement shared among these integrated design optimisation packages is that users need to provide the optimisation modules with programmatic access to the modelling and analysis tools that supply the objective functions. Nevertheless, such modelling and analysis tools are in many cases developed with different technologies and therefore require different software interfaces. For example, there exist different CAD packages, finite element analysis (FEA) and computational fluid dynamics (CFD) codes. It is therefore necessary to develop wrappers for these packages to be used in integrated environments mentioned earlier.

A variety of technologies have been applied to support the interactions between incompatible software components. The most commonly used method is to communicate via data files, which relies on shared data types and formats, or specially developed parsers to interpret the input/output files. There are sometimes standards available to help in the exchange of such files. For example, the Standard for the Exchange of Product Model Data (STEP) [5] was made to facilitate product data exchanges. Another approach to integration is based on common object interface technologies, such as CORBA [7], in which function calls to the components are carried out as standard remote procedure calls (RPC). These technologies have so far only achieved limited success. While the use of exchange data files can be seen as a generic approach, the lack of standard formats in native data description and semantic descriptions of the file content means that extra layers of processing are required almost every time a new component is introduced. And it lacks the capability to deal with fault tolerance in a consistent manner. As for the common interface technologies, the use of vendor-specific standards and protocols makes it very difficult to achieve wider interoperability. And the use of RPC risks tight coupling that makes the system less flexible and scalable. Furthermore, since most of these technologies are not firewall-friendly, it is not easy for them to be applied in a wider distributed environment such as the Grid.

Recent advancement of Grid computing technologies has provided the incentive for a new solution to integrate optimisation in engineering design. The primary target of Grid technologies is to enable large-scale, dynamic collaboration among participants from highly heterogeneous

and distributed computing environments. A number of attempts have been made to achieve this before Web service technologies were introduced. It is now widely recognized that a service-oriented approach, by which all resources on the Grid are viewed as services built on standard interface definition and invocation mechanisms, is able to provide the desired interoperability and facilitate collaboration. In adopting service-orientation in Grid computing, Web services are extended to the more sophisticated Grid services, such as those proposed in the Open Grid Service Architecture (OGSA) [8] and the associated Open Grid Service Infrastructure (OGSI) [9].

The idea of presenting numerical optimisation technologies as Grid services arose from our efforts to adopt service-oriented Grid technologies for engineering design optimisation [10]. It offers a generic and extensible framework to address the integration issues by completely decoupling the optimisation modules from the other software components. The optimisation codes, regardless of what programming language they are written in or what the platform they run on, are encapsulated into standard Grid services that are universally accessible. The tightly-coupled programmatic links between the optimisation modules and the modelling codes that used to be required for integration are replaced with loosely-coupled, standard based message level interactions. It therefore becomes easier to adopt in one engineering design system a number of different optimisation technologies, or to apply one optimisation method to a variety of design problems. Furthermore, the use of standard communication protocols for Grid services makes it firewall friendly and therefore appropriate for a distributed, cross-institutional design environment.

The focus of the work presented in this paper is to expose optimisation services in a flexible, generic interface that can be easily integrated into various environments and used to compose different optimisation workflows. The rest of this paper is organised as follows: In section 2, we review the related work by examining the advantages and disadvantages of these technologies and tools. Section 3 discusses the motivation and concept of Reverse Communication interface. In section 4, details are provided on the design of the optimisation grid services, which is illustrated in section 5 with two sample implementation of the services based on different optimisation algorithms. In section 6 we demonstrate the optimisation services with their applications in solving different engineering design optimisation problems, and discuss the service performance issues based on the experiences. We draw our conclusions in section 7.

2. Related Works

A number of attempts have been made to enable numerical optimisation in distributed environments. One of the most notable ones is the NEOS project [11]. However, it requires that the design problems be formulated in the AMPL languages [12] and submitted to the server for execution. Therefore it is not applicable to cases where the objective functions and constraints need to be computed using commercial or proprietary codes.

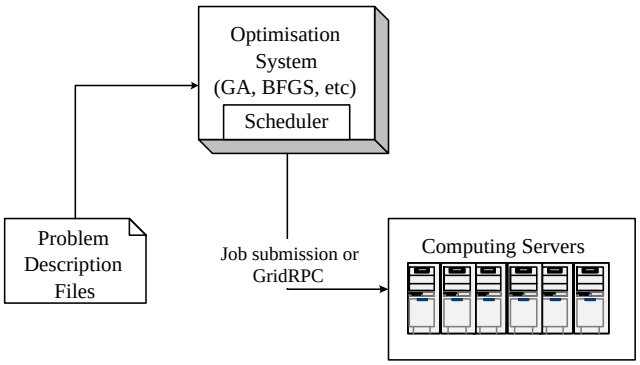


Figure 1. Architecture of a Typical Optimisation System using Forward Communication

Other related work such as iSight, FIPER, ModelCenter and Nimrod/G [13] [14] mentioned earlier adopt a different approach: optimisation is often an inherent part of an integrated environment, in which modelling and analysis packages are often integrated using CORBA, RMI or other RPC technologies. In such systems the optimisation logic is usually closely coupled with the job submission and scheduling functions. Apart from the numerical optimisation search, the optimiser is also responsible for passing forward information on where the next set of design points should be, and controlling the evaluation of objective functions through job submission systems such as Netsolve [15], Globus [16] or GridRPC [17]. This type of system is often called "Forward Communication". The major disadvantage of such approach lies in the following aspects: firstly, it is difficult to make use of the optimisation algorithms from outside the integrated system, and to incorporate new optimisation algorithms into the system by end users; secondly, it is also difficult to replace the job submission and scheduling components; and finally, users are restricted to formulate their problems within the restrictions of the system. Figure 1 shows a typical architecture of such systems.

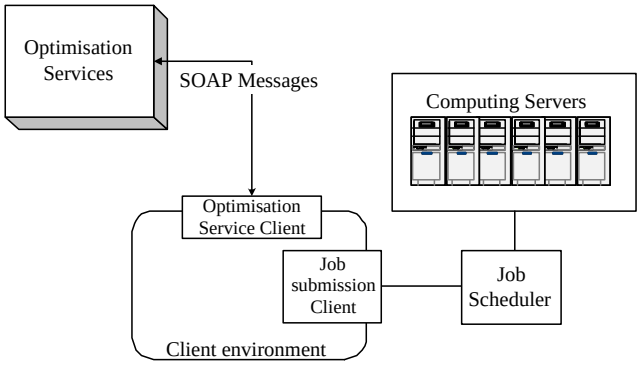


Figure 2. Architecture of Service-Oriented Optimisation System

In systems as illustrated in Figure 1, users are required to wrap their objective functions in a prescribed format and language for the optimisation system to submit to computing resources. It lacks the flexibility that is desired for accessing the optimisation algorithms from outside the integrated environments, which makes it infeasible to share optimisation methods among heterogeneous design environments distributed across multiple administrative domains. Moreover, the proprietary interfaces used by these systems also decide that users will have to develop individual interfaces for different packages.

Our vision in solving these problems is to embrace recent developments in Web services and Grid technologies so as to deliver highly scalable and flexible optimisation services to the design environments using a generic, loosely-coupled interface. This approach makes numerical optimisation a stand alone module that can be accessed from various programming languages, PSEs and middleware without knowledge of the implementation details. Figure 2 illustrates the architecture of our proposed system based on the use of "Reverse Communication" in interface design, which will be discussed in the next section. Users can communicate with the optimisation services via standard SOAP messages using client tools in a design environment of their own choice, for example, Matlab [18] or Jython [19].

3. Service-Oriented Numerical Optimisation with Reverse Communication

In this section, we explain the rationale behind the use of the Reverse Communication interface [20] [21] which makes it possible to apply service-orientation to present numerical optimisation algorithms.

First it helps to have a brief review of the typical structure of common numerical optimisation packages and their limitations. Numerical optimisation is an iterative process whereby modelling and analysis codes are exploited to produce improved designs. The modelling and analysis codes are used to calculate the values of functions that, in some sense, characterise the performance of the design. These "measures-of-merit" or "objective" functions and constraints are often calculated using high-fidelity analysis codes in fields such as computational structural mechanics and computational fluid dynamics, and normally require high performance computing facilities. A requirement shared among most existing optimisation packages is that users need to provide the optimisers with a "hook" to users' analysis codes. While this can be implemented in various ways such as shared data files, message passing and direct code linkage, a common feature can be observed: the process is driven by the optimiser in a Forward Communication style. As a result, users are required to conform to the strict interface requirements when integrating their modelling and analysis codes into the optimisation framework. Figure 3 shows a typical structure in which users have to pass a function pointer to the optimiser.

In addition, in a forward communication optimisation system, the optimiser not only process the application logics, but also have to manage job scheduling, computing resources allocation and visualisation. Such system structure is only applicable to a single package, or when a global standard exists. As it is usually unlikely to have such standards, a more flexible model is needed to carry out the task of integrating heterogeneous packages into the optimisation process.

We apply a service-oriented structure to the design optimisation system, by which the optimisation codes are presented as independent components with standard interfaces. Being Grid services, the optimisers are driven by the optimisation process using standard XML based messages, and are no longer responsible for the running of the other parts of the process. It is therefore possible to integrate optimisation codes with different design systems.

Service-oriented numerical optimisation is only feasible when using the Reverse Communication mechanism, which establishes a server/client relationship between the optimiser and its users. The idea of Reverse Communication is not new. It has seen limited use in some numerical libraries [20] and optimisation implementations [21]. It provides a flexible measure to couple the optimisers and analysis code. Reverse Communication works by iteratively calling the optimiser with the aid of a flag, which indicates the actions the user requires, such as process initialisation. Each time the optimiser returns a result, it will indicate to the user what operations must is expected next and what informa-

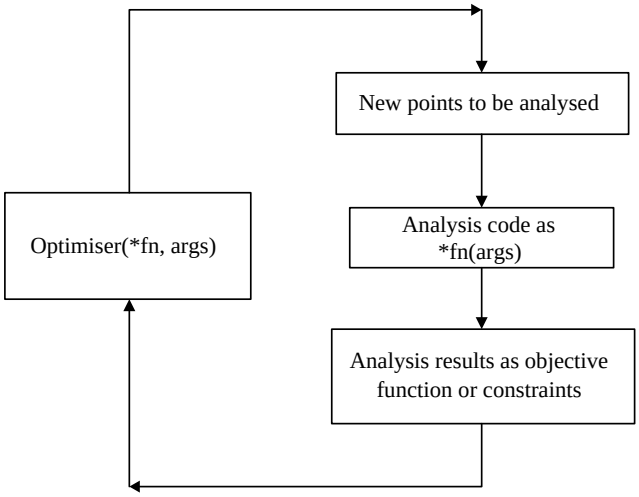


Figure 3. A typical structure for numerical optimisation with the analysis code specified as a function pointer($*f_n(args)$)

tion is required to be passed back. It is essentially a client-server style of interactions between the optimiser and the user supplied analysis code, which provides maximum flexibility for the implementation of the iterative process. The user has complete control over when and where to request the service (i.e., the optimiser). And it is possible to provide multiple client tools for different target design environments.

This type of interface avoids the need for tight coupling between the optimisers and the user's analysis codes. Therefore it provides a general infrastructure for implementing various optimisation algorithms as server-side services. The concept of implementing optimisers as services is well-matched to the typical scenarios in a Grid computing environment (GCE), in which, the ability to perform compositional modelling is the key for building complex modelling capabilities. The implementation of optimisation as a Grid service also makes it easy to plug in new functionalities such as parallelisation and scheduling of computational jobs and archiving of computational results, which usually rely on the implementation within the optimisers themselves. Other benefits include: it becomes unnecessary to keep alive the communication channel between the optimiser and analysis codes on the computational servers; it is much easier to implement parallel algorithms in user's codes; it is easier to add or replace scheduling algorithms for multiple computing jobs or exploit best-of-breed third party systems; and optimisation can be implemented to deal with multiple tasks simultaneously.

A problem of applying Reverse Communication, nevertheless, is that it may require the re-engineering of the optimisation code in a way that might only be possible with the availability of source code.

A generic Reverse Communication interface for an optimiser can be represented as follows.

$$ret = opt(task, \boldsymbol{x}, f, \boldsymbol{g}, \boldsymbol{p}) \tag{1}$$

where, on entry, *task* is the character string indicating the task required to the optimiser, $\boldsymbol{x}$ represents the current values of the design vector, f denotes the current function value, $\boldsymbol{g}$ denotes the gradient vector at the current point, and $\boldsymbol{p}$ denotes the control parameters provided by the user. The return value *ret* indicate the status of the call to the optimiser.

Table I. List of Service Tasks on Invocation

Task	Description
"start"	Initialise optimisation process from the value provided in $\boldsymbol{x}$.
"new"	Objective function value provided at point $\boldsymbol{x}$.
"grad"	Gradient vector provided at point $\boldsymbol{x}$.

On return, *task* contains the next task required by the optimiser, it can be the following.

Table II. List of Service Status on Return

Task	Description
"new"	Function value required at the point $\boldsymbol{x}$, which has been modified by the optimiser *opt*.
"grad"	Gradient vectors required at the point $\boldsymbol{x}$.
"stop"	Solution converged with results stored in $\boldsymbol{x}$.
"fail"	Optimiser fails to converge, adjust the convergence criteria.

The workflow of the optimisation process using service-orientation is illustrated in Figure 4, in which the invocations of the service are controlled by the process, instead of the other way around. Details on the design and implementation of optimisation as Grid services in this way are discussed in the following sections.

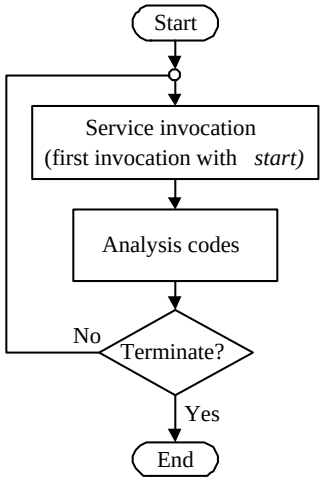


Figure 4. Illustrative Flowchart using Optimisation Grid Services

4. Design of the Optimisation Grid Services

This section discusses the design of the optimisation services which aim to offer a variety of numerical optimisation methods. There are four major issues: the design of the service architecture, the design of a generic optimisation service class, the integration of native, proprietary numerical optimisation codes, and security management for the optimisation services.

4.1. Architecture of the Optimisation Service

Numerical optimisation services are stateful Grid services. A complete optimisation process usually involves multiple interactions with the optimisation service. The service needs to maintain information concerning the optimisation process, including the choice of optimisation method, bounds of design variables, control parameters of the method, and search history of the optimisation process between service invocations. To ensure service performance, the services need to be able to restore states quickly once a new request is received. In the case of a system failure, the services should be capable of recovering previous states when the system is restored.

When designing the architecture of the optimisation services, the primary concern is whether it can provide a simple and efficient model for service state management. Traditional Web services are stateless.

The standard Web services specifications do not bear any notion of application state. There are mainly two different approaches for state management of Web services: the transient Grid service model [9] and the use of operation context.

The transient grid service model is based on OGSA and is technically specified in OGSI. The basic idea is to create service 'instances' that encapsulate all operation-specific state information. Similar to distributed objects, the service instances are instantiated through a factory service and are uniquely identified using Grid Service Handlers (GSH) [8]. All service interactions for a particular operation are carried out towards the same service instance. The instances only exist for the lifetime of the operations and are destroyed once the operations are finished.

Instead of introducing an additional infrastructure like OGSI, the operation context based approach attempts to provide support for stateful service interactions within the Web services framework, using specifications such as WS-Context [22]. Operation states are encapsulated in XML based entities called "context". Each service message for a particular operation contains in its SOAP header a context, or a URI link to it. The target service retrieves state information through the context to handles the requests [23].

Between the two approaches, the OGSI transient grid service model has been selected as the basic infrastructure for our numerical optimisation service for two reasons. Firstly, it can be far superior in system performance to the context based approach. A complete optimisation procedure may involve hundreds or thousands of search steps. When using context, all state information needs to be loaded to the system every time a request comes in. Compared to the transient service model, by which the service instance is always maintained in the system and can be accessed immediately, the accumulative increase on system load can be significant. Secondly, state management with the transient grid service model is less complicated. Since all states are maintained by the service instances, it is not necessary to introduce additional specifications or to define new data structures. And, unlike the context based approach, the service clients do not participate in state management. This simplifies the integration with the optimisation services.

The design of the optimisation service architecture based on the transient grid service model is shown in Figure 5.

The optimisation service is composed of both stateful and stateless parts. The stateful part of the service is the instance object that is responsible for carrying out service operations and keeping the service state data. Created from the optimisation service class, the instance objects are maintained in an object repository within the OGSI Container for the lifetime of the transient service. And each instance object

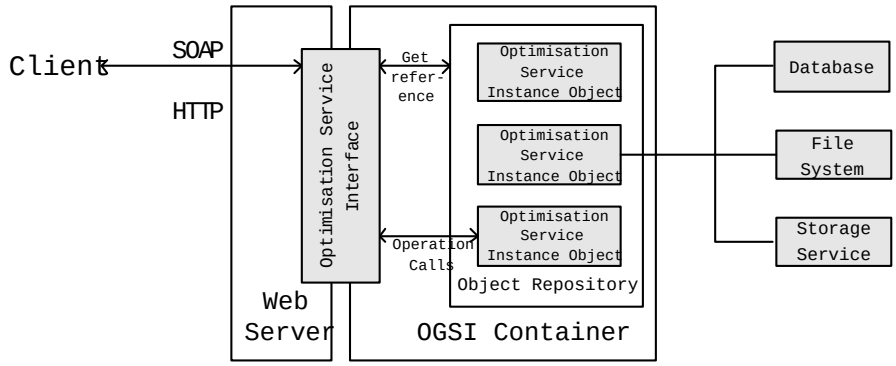


Figure 5. Architecture of the Optimisation Grid Service

registers with the container using a unique ID that can used to locate the object in the repository. The service interface is the stateless part of the service, which serves as a bridge between the web server and the OGSI container. It interfaces to the optimisation service by handling the standard SOAP protocol [27] and associated protocols for security and data transmission. Each time the service is invoked, the service interface processes the request message, interacts with the OGSI Container to locate the target instance object based on the Grid service handle (GSH) provided by the client, starts the operation on the instance object, and finally send the results back to the client in SOAP messages.

The OGSI Container usually maintains the service instance objects in the system memory so as to minimise response time. It is nevertheless not reliable since all the service states can be lost when there is a system failure. Consequently, our design includes permanent storage mechanisms such as databases, file systems, and data (Grid) services [28], where data from the service instance objects can be serialised. The backup operation can be carried out automatically by the optimisation service, or when the client requires.

4.2. Design of the Optimisation Service Class

The service instance objects constitute the principal part of an optimisation service. Each object is an instance of an optimisation service class, which defines the data structure and behaviour of the service. In order to facilitate and standardise the service development, a predefined abstract class named OptimisationGridServiceSkeleton is designed to be implemented and extended by new optimisation service classes. It provides a generic abstraction of optimisation services by

defining standard data fields and interfaces shared by most optimisation methods. In addition, the abstract class also implements general Grid services features, which are required for the optimisation services to be hosted by the selected OGSI environment.

To support optimisation in the Reverse Communication style, the OptimisationGridServiceSkeleton class includes the following data fields:

- ***methodType***, an integer, which indicates the category that the optimisation method belongs to.
- ***optimisationMethod***, an integer, which indicates the optimisation method current service instance employs. It is only used when the service provides multiple optimisation methods.
- ***designVectorSize***, an integer, which indicates the size of the design vector.
- ***designVector***, a double array, which stores the current value of design variables
- ***boundSet***, a boolean array, which indicates if bounds exist on design variables
- ***bounds***, a double array, which can be used to set the bounds on design variables for optimisation methods with simple bounds on design variables.
- ***gradientSet***, a boolean variable, which indicates whether gradient information is used by the method
- ***gradientVector***, a double array, which stores the current value of the gradient vector. It is used by gradient-based optimisation routines such as hill climbing
- ***hessianSet***, a boolean variable, which indicates whether the Hessian is used by the methods
- ***hessianMatrix***, a double array, which stores the value of the Hessian matrix, it is only used by methods which use second order derivatives
- ***scaleVector***, a double array, which stores the current value of the scale vector, and can be set by the client
- ***objFuncVector***, a double array, which stores the current value of the objective functions.

- ***constraintsVector***, a double vector, which stores the current value of all constraints if these exist
- ***iterationsRequested***, an integer, which stores the number of iterations requested by the user.
- ***optimisationHistory***, which logs each step of the optimisation, including values of the design variables, the objective function values, constraints, and the control parameter values.
- ***serviceOwner***, which records the identity of the owner of the current service instance. The identity can be a username, or the subject line of a digital certificate.

The abstract class also defines the following operations:

- ***Optimise***. This operation is called to start or continue the optimisation. Input and output parameters can be defined with reference to the tasks listed in Table I and II in the previous section. All optimisation services are required to implement this method.
- ***Reset***. This operation reloads the initial service state that is set up when the service instance is created, or restores a previous service state of the service instance. It can be implemented to enable customized service restart.
- ***SetServiceParameter*** and ***GetServiceParameter***. These two methods allow users to access control parameters of the optimisation process. As control parameters of different optimisation methods are diverse in their formats, no specific data field is defined for them in the abstract class.
- ***SaveServiceState***. This operation allows users to explicitly require the current service state to be saved. An index symbolising the saved state is usually returned, which can later be used by the RESET method to find the required state information.

In addition, the abstract class also comes with a default constructor, which can be used to initialise the service instance object when it is created. It allows users to specify the choice of optimisation method and the size of the design vector. It also initialises the pre-defined vectors, as well as the service history log. When required, customised constructors can also be made to extend or replace the default constructor.

The use of the abstract class for services with different optimisation methods is demonstrated in the next section with our sample optimisation services.

4.3. Integration of Legacy Numerical Optimisation Code

An important issue in the development of the optimisation services is how to integrate the target optimisation methods with the services, using advanced development environments and tools. Most existing optimisation methods are written in native programming languages such as FORTRAN and C/C++ that are different from those used to build Grid/Web services. Two approaches can be applied to seamlessly integrate the service programming environment and the optimisation codes, as shown in Figure 6.

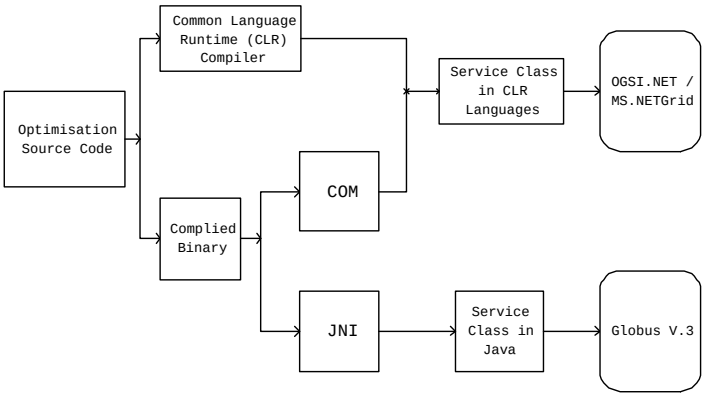


Figure 6. Encapsulating Existing Optimisation Code into Service Classes

The first one makes use of the Common Language Runtime (CLR) [29], which provide multi-language support and seamless integration of code written in different programming languages, such as FORTRAN, C/C++, VB and C#. It is therefore possible to have existing source code compiled into CLR libraries, and be accessed through CLR by service classes written in different languages. The CLR approach requires access to the source code of the optimisation methods. In many cases, however, the optimisation methods are only available as binary libraries. The second approach addresses this problem by using native interface technologies such as Java Native Interface (JNI) [30] and COM [31] in the service classes to access the compiled binaries.

Choice between these two approaches can be made based on performance, source code availability, and most importantly the selection of the service hosting environment. When .NET based Grid service environments such as OGSI.NET [26] and MS.NETGrid [25] are selected, legacy optimisation codes can be integrated using either CLR or COM. On the other hand, if the optimisation services are to be deployed over

Java based Grid service environments such as Globus v.3 [24], JNI will have to be applied.

In the next section, the use of CLR and COM are demonstrated through our sample optimisation services. Examples for the use of JNI can be found in [33].

4.4. Security Management

Security management for the optimisation grid services are mainly concerned with the following three issues: message integrity, privacy, and service ownership. The service needs to make sure that the messages it receives have not been damaged or altered, the content of the messages are only accessible by the sender and the service instance, and only the requests from the owner of the service instance are accepted.

For Web services, security mechanisms are available at both the communication level and the message level to ensure message integrity and privacy. Deploying security over the communication layer risks the tight coupling of the optimisation services to a particular protocol, and thus makes the solution less generic. We therefore deploy WS-Security [34], an XML-based message level security protocol, to address the security requirements of the optimisation services. WS-Security specifies how XML signatures should be created to guarantee the authenticity and integrity of the SOAP messages, and how the messages can be encrypted to maintain the privacy. The X.509 v3 certificates have been selected as the security token for the optimisation service. Details on how WS-Security and X.509 work can be found in [35].

5. Implementations of the Optimisation Grid Services

In this section we demonstrate how the optimisation grid services can be implemented to provide various numerical optimisation methods. Two typical examples have been selected, including a gradient based optimisation routine from the PORT library [36], and a genetic algorithm (GA) from the OPTIONS system [40]. Both services have been constructed under .NET based OGSI platforms, with the gradient based service on MS.NETGrid, and the GA service on OGSI.NET.

5.1. Building the Gradient Optimisation Service

As part of this work we have developed a demonstrative optimisation service based on optimisation routines from the PORT Mathematical Subroutine Library [36]. The PORT library contains routines that implement Reverse Communication versions of optimisation algorithms

for general constrained / unconstrained minimisation. Source codes of these routines are available in FORTRAN 77. With the help of the CLR compiler for FORTRAN, we were able to convert these into libraries accessible to the service class written in C#, and to construct the service based on the .NET Grid service hosting environment.

The following optimisation routines are encapsulated in the sample service:

- ***RMNF*** and ***RMNFB***, which minimise general constrained / unconstrained objective functions using finite-difference gradients and secant Hessian approximations.
- ***RMNG*** and ***RMNGB***, which minimise general constrained / unconstrained objective functions using double-dogleg/BFGS steps.
- ***RMNH*** and ***RMNHB***, which minimise general constrained / unconstrained objective functions using a Hessian matrix provided by the caller.

Each instance of the gradient service is bound to a particular optimisation method. When starting an optimisation process, the message that requests the creation of a new service instance specifies the target optimisation method, the size of the design vector, and the boundaries of the search parameter space. Data fields within the service instance object will be initialised accordingly. Figure 7 shows the request message to create an instance of the gradient optimisation service that uses RMNF. It also specifies that the optimisation is to be carried out in an unconstrained 3-dimension parameter space.

The gradient service implements all methods defined in the OptimisationGridServiceSkeleton abstract class. In particular, the SetServiceParameter and GetServiceParameter methods have been implemented to handle two extra data fields that are added to the service class to control the optimisation process:

- ***taskControl***, an integer vector, which contains control variables for the optimisation.
- ***convergence***, a double vector, which stores convergence criteria for the problem.

The service has been deployed on the MS.NETGrid platform, which integrates with IIS [37] and ASP.NET [38]. It is therefore feasible to implement the service security management by directly exploiting supports for WS-Security from the Web services enhancement package [39].

```
<?xml version="1.0" encoding="utf-8" ?>
  <soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
+ <soap:Header>
  <soap:Body wsu:Id="Id-7946865e-472c-4b9d-b2cc-2b7df53e6ead"
       xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
    <createService xmlns="http://ogsa.gridforum.org/factory">
      <creation xmlns="http://www.gridforum.org/namespaces/2002/10/
                              gridServices">
        <serviceParameters
           xmlns="http://www.gridforum.org/namespaces/2003/03/OGSI">
          <ServiceSettings xmlns="http://www.geodise.org/optimisation">
            <OptimisationMethod>    RMNF</OptimisationMethod>
            <DesignVectorSize>   3</DesignVectorSize>
          </ServiceSettings>
        </serviceParameters>
      </creation>
    </createService>
  </soap:Body>
</soap:Envelope>
```

Figure 7. Request Message to Instantiate the Optimisation Grid Service

5.2. Building the GA Grid Service

The GA grid service provides generic access to well-established implementation of a genetic algorithm from the design exploration system OPTIONS [40], therefore allowing users to apply the algorithm in various computing environments of user's choice.

The service has been built based on existing compiled Win32 libraries, which are not directly accessible by the .NET based service implementation. To solve this problem, a wrapper library has been built using the Salford FTN95 compiler. It interacts with the Win32 libraries using COM, while interfacing itself as a .NET library.

The workflow for a typical conventional genetic algorithm is shown in Figure 8(a). To implement it as a service, it is necessary to convert it to the workflow shown in Figure 8(b), which effectively divides each iteration into two steps after the initialisation of the first population and evaluation of the fitness functions. The first step in one generation is to request the design variables for the next generation based on the GA's evolution mechanisms (selection, crossover and mutation), in the second step, the values of the fitness functions are sent back to the Grid service. The main difference between these two workflows is exactly where the GA operators are applied.

The converted GA provides the following three routines that have been applied in the implementation of the GA optimisation service.

- ***ga_optdbs***, which initialise the internal data structure used by the optimisation package.

```
gen=1
Pop(gen) = randomly generate first generation
evaluate fitness of all individuals in the population
while (termination condition = false)
        gen = gen + 1
        apply generic operators to pop(gen)
        evaluate fitness of the population
end
```

(a) Conventional GA

```
Get the initial generation from GA service (ga_next (first =1))
while (termination condition = false)
        evaluate fitness of the population
        notify the GA service on the fitness values (ga_objs)
        get the next generation from GA service (ga_next(first=0))
end
```

(b) Re-engineered service-oriented GA

Figure 8. A Comparison of Conventional and Converted GAs

- ***ga_next***, which evolves the GA by one generation.
- ***ga_objs***, which returns objective functions and constraints for the current population to the GA.

The ga_optdbs routine is used by the service constructor to initialise the service instance. It is also used in the implementation of the RESET service method when a restart of the optimisation process is required. The ga_objs and ga_next routines are used in the implementation of the Optimise service method.

The GA optimisation service has been deployed on the OGSI.NET platform, which provides a robust attribute-based programming model for service development. OGSI.NET also provides implementation of WS-Security, as well as mechanisms to declare the service's security policy.

6. Experiences and Discussion

In this section we demonstrate the use of the optimisation services implemented in the previous section with concrete engineering design

optimisation problems. The experiences show that running optimisation as grid services can provide a successful solution to the seamless integration of numerical optimisation methods for engineering design optimisation. We also discuss performance considerations for our service-oriented approach based on the demonstrations.

Both engineering design optimisation problems demonstrated here are orchestrated using the Matlab scripting language, which is one of the most commonly used scripting environments to engineers. To access the optimisation services from Matlab, a Java based service client has been constructed to handle the interactions and the client-side security management. Since Matlab runs its own Java Virtual Machine, the service client can be used the same way as common Matlab commands.

6.1. Airfoil Optimisation Based on Orthogonal Basis Functions

Shape optimisation of airfoils has been extensively studied in the aerospace industry, and was chosen here to demonstrate the effective use of our optimisation services in a concrete engineering design study. The airfoil geometry is defined as a linear combination of basis functions, as shown in Equation 5. More details of the problem can be found in [41].

$$f = \sum_{i=1}^{n} w_i f_i \tag{2}$$

The coefficients of these basis functions are specified as design variables, and the geometry is modelled in the CAD package ProEngineer [42]. The lift/drag ratio is designated as the objective function and computed using the computational fluid dynamics code Fluent [43]. In order to overcome the problem of the high computational cost of the CFD analysis, a combination of design of experiments and response surface modelling methods is used to build a surrogate model, and then the search is carried out on the response surface model. Detailed discussion on how these methods can be employed can be found in a number of papers, for example, [45]. The response surface model for a two-variable problem (the first two coefficients in the definition) and corresponding search path is shown in Figure 9. The original and final airfoil sections are shown in Figure 10 along with the plots of pressure around these sections.

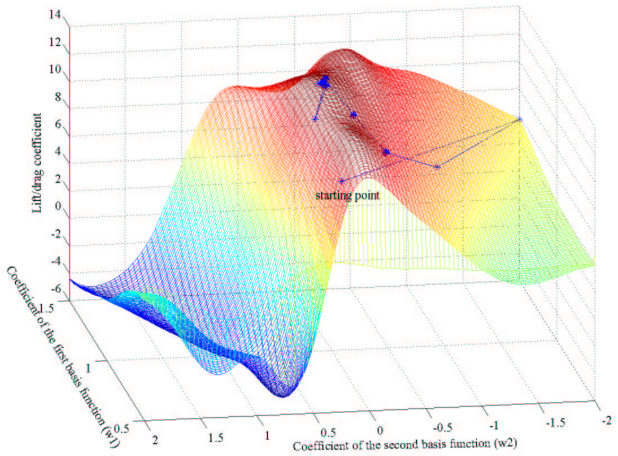
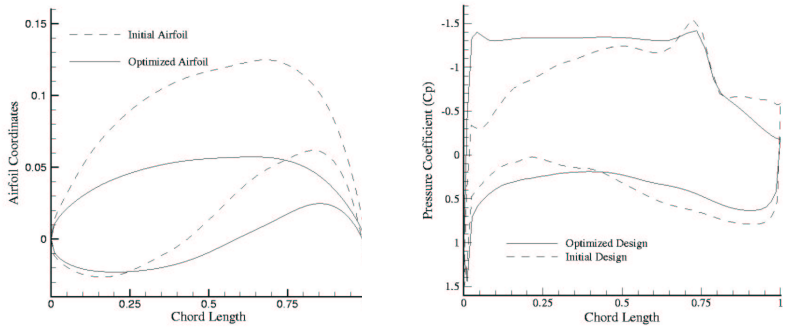


Figure 9. Response surface model and search path for the two-variable airfoil problem with the starting point at $w_1 = 0.6; w_2 = 0.1815; l/d = 9.9238$. Final point found at $w1 = 1.14; w2 = -0.031; l/d = 12.2636$

The Matlab script used to operate the sample service for the airfoil optimisation is shown in Figure 11. It shows how the optimisation services can be seamlessly integrated into customised design environments. Scripts marked out with bold font indicate interactions with the services.



a. Airfoil Shapes b. Pressure Coefficient

Figure 10. Result of the Airfoil Optimisation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Java client need to be in the classpath
import optimisationGS_javaclient.*;

% create Client object with specified problem dimension
service = Client(2, false);
optimiser=Optimiser.RMNFB;

% set boundary
boundary=[lvars(1), uvars(1), lvars(2), uvars(2)];
service.setBoundary(boundary);
service.createService(optimiser);

d=[1,1];  % scaling point
service.SetServiceParameter(ServiceParameter.SCALE, d);
service.SetSingleParameter(ServiceParameter.CONVERGENCE, 34, 10.0);

output = OptimisationOutput;
% first call to service
output = service.Optimise(ServiceTask.START, x, [], [], obj);
x=output.DesignVector;

nfcall =5;
toobig =2;
% main optimisation loop
nloop = 1;
while ( ClientTask.NEW.toString == output.task.toString)

    nf = service.GetSingleIntParameter(ServiceParameter.TASKCONTROL, nfcall);
    orthfoilRSM.vars=x;
    opt.Search(orthfoilRSM);
    obj = -orthfoilRSM.objfn;

    objhis(nloop)=double(obj);
    if nf <= 0

        service.SetSingleParameter(ServiceParameter.TASKCONTROL, (toobig-1), 1)   ;
    end
    output= service.Optimise(ServiceTask.FUNC, x, [], [], obj);
    x=output.DesignVector;
    nloop= nloop+1;
    xhis(:,nloop)=double(x);
end
service.destroy ;
```

Figure 11. Matlab Script for Airfoil Optimisation on the Sample Service

6.2. Optimisation with the GA Grid Service

The implemented GA optimisation service is applied to another more complicated engineering design optimisation problem, in which parameters in a parametric engine nacelle geometry are modified to study their effect on the aerodynamic performance of the design. A fully parametric model was built in ProEngineer, and then linked to the meshing package Gambit and the solver Fluent to calculate the aerodynamic performance in terms of the total pressure recovery at the fanface. The GA service is then used to carry out the optimisation search on the model. The whole process is automated within the Matlab environment.

The response surface model of the objective function (total pressure recovery) is shown in Figure 12, together with the best point found using the GA service.

6.3. Discussion of Performance

The impact on optimisation process performance brought by our service-oriented approach, i.e. the prolonged optimisation search cycles, can be

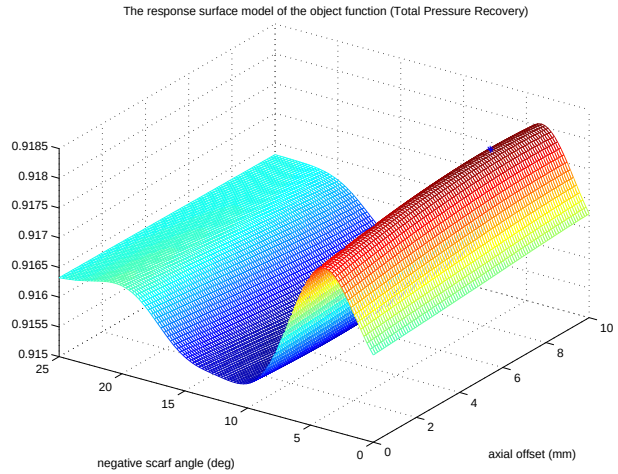


Figure 12. Response surface model of the objective function (Total Pressure Recovery, *: highlighted maximum)

ascribed to two reasons. The first one is the use of network communication for interactions between system components. In order to provide generic and firewall-friendly access to the optimisation services, all interactions are carried out via HTTP, which is inferior in performance to other low-level protocols. The second reason is the message overhead brought in by the use of XML and SOAP. For example, for the first demonstration, to deliver the input data (about 13 bytes), a message of 374 bytes is sent to the service. The overhead is even more significant when security features like XML Signature and XML Encryption are applied.

Nevertheless, it should be pointed out that in practical application of the optimisation services, the performance penalty brought by the services is relatively insignificant from the users' perspective. Records from the first demonstration show that an interaction with the optimisation service takes on average 0.04687 seconds, while each calculation of the objective function takes approximately 1800 seconds. It is obvious that performance of the optimiser is not critical with regard to the entire optimisation process, and the performance impact can actually be ignored.

7. Conclusions and Future Developments

In this paper, we have introduced a generic, service-oriented approach for applying numerical optimisation algorithms in engineering design.

Presenting numerical optimisation as Grid services makes it possible to integrate desired optimisation methods with target engineering design environment, regardless of the differences in programming languages or executing environments. It also makes it feasible to build engineering design systems in a truly distributed environment.

The optimisation service relies on the Reverse Communication model, which makes it possible for optimisation codes to be implemented as services. The architecture of the optimisation service was designed to provide strong support for management of the optimisation states that are essential to Reverse Communication. Issues including the design of the service class, integration of existing optimisation packages, and security management of the services were discussed in detail to provide guidance for implementing optimisation as grid services. We demonstrated our approach by showing optimisation services built from existing numerical optimisation packages with different levels of accessibility. Results of the applications of the services in solving real engineering design optimisation problems were presented to validate the approach.

Future work will focus on the adaptation of the optimisation services alongside the development of grid service infrastructures within the Grid computing community. Currently, the service has been designed and developed based on the OGSI. With the introduction of the WS-Resource Framework (WSRF) [46], which is a refactoring and evolution of OGSI towards open standard Web services, the optimisation services will be revised to adapt to new service hosting environments. However, as explained in [47], the transition from OGSI to WSRF is not revolutionary but evolutionary. Therefore the basic architecture of the optimisation services and the way the services operate will not be affected. Changes will mainly be applied to the implementation details.

In addition, we will also attempt to apply our approach to more optimisation algorithms. Application of the optimisation services in areas other than engineering design, such as structural problems and photonic devices, will also be investigated.

Acknowledgements

This work is supported by UK e-Science Pilot Project "Grid-Enabled Optimisation and Design Search for Engineering (Geodise)" (UK EPSRC GR/R67705/01). We also gratefully acknowledge support from Microsoft.

References

[1] Papalambros, P.Y., and Wilde, D.J. Principle of Optimal Design. ISBN 0-521-62727-2. Cambridge University Press, 2000.

[2] Baalbergen, E.H., and Van der Ven, H. SPINEware: A framework for user-oriented and tailorable metacomputers, NLR-TP-98463, National Aerospace Laboratory, NLR, 1998

[3] Interdigitation for Effective Design Space Exploration Using iSIGHT. Journal of Structural and Multidisciplinary Optimization, Vol. 23, No. 2, pp. 111-126, 2002

[4] ModelCenter. http://www.phoenix-int.com/2004/

[5] STEP. http://www.steptool.com/, 2004

[6] Rohl, P.J., Kolonay, R.M., Irani, R.K., Sobolewski, M., Kao, K., and Bailey, M.W. A Federated Intelligent Product Environment. 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, CA, September 6-8, 2000

[7] Fintan Bolton. Pure CORBA. ISBN 0672318121. Sams, 2001.

[8] Foster, I., Kesselman, C., Nick, J.M., and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[9] Tuecke, S. et. al. Open Grid Service Infrastructure (OGSI) Version 1.0. Global Grid Forum. http://www.gridforum.org/ogsi-wg/

[10] Cox, S.J, Chen, L, Campobasso, S, Duta, M.H, Eres, M.H, Giles, M.B, Goble, C, Jiao, Z, Keane, A.K, Pound, G.E, Roberts, A, Shadbolt, N.R, Tao, F, Wason, J.L, Xu, F. (2002) Grid Enabled Optimisation and Design Search (GEODISE). UK e-Science All Hands, Sheffield, 2-4 Sept 2002

[11] Michael C. Ferris, Michael P. Mesnier, and Jorge J. More, NEOS and Condor: Solving Optimization Problems Over the Internet, ACM Transactions on Mathematical Software, Vol. 26, No.1, March 2000, pp 1-18.

[12] Fourer, R., Gay, D.M., and Kernighan, B.W. AMPL: A Modeling Language for Mathematical Programming,Duxbury Press, Brooks/Cole Publishing Company, 2002

[13] Abramson, D, Lewis, A. and Peachy, T., "Nimrod/O: A Tool for Automatic Design Optimization", The 4th International Conference on Algorithms & Architectures for Parallel Processing (ICA3PP 2000), Hong Kong, 11 - 13 December 2000

[14] Shields, M.S., Rana, O.F., Walker, D.W., Li, M., and Golby, D. A Java/CORBA-Based Visual Program Composition Environment for PSEs, Concurrency: Practice and Experience, Vol. 12, pages 687-704, 2000.

[15] Arnold, D., Dongarra, J. "The NetSolve Environment: Progressing Towards the Seamless Grid," 2000 International Conference on Parallel Processing (ICPP-2000), Toronto, Canada, August 21-24, 2000.

[16] Foster, I., Kesselman, C., Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[17] Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C., and Casanova, H. GridRPC: A Remote Procedure Call API for Grid Computing. http://www.eece.unm.edu/ apm/docs/APM_GridRPC_0702.pdf

[18] Hunt, B., Lipsman, R., and Rosenberg, J. A Guide to MATLAB: for Beginners and Experienced Users. ISBN 0521-00859-X. Cambridge University Press, 2001.

[19] Bill, R. Jython for Java Programmers. ISBN0735711119. Pearson Education, 2001.

[20] Dongarra, J., Eijkhout, V., and Kalhan, A. Reverse Communication Interface for Linear Algebra Templates for Iterative Methods, UT, CS-95-291, May, 1995

[21] Gay, D.M. Usage Summary for Selected Optimisation Routines. http://netlib.bell-labs.com/cm/cs/cstr/153.pdf

[22] OASIS(WS-CAF), Web services Context (WS-CTX). www.iona.com/devcenter/standards/WS-CAF/WSCTX.pdf

[23] Parastatidis, S., Webber, J., Watson, P., and Rischbeck, T. A Grid Application Framework based on Web Services Specifications and Practices. http://www.neresc.ac.uk/projects/gaf

[24] The Globus Alliance. Globus Toolkit 3.2 Documentation. http://www-unix.globus.org/toolkit/docs/3.2/index.html

[25] The MS.NETGrid Project. http://www.epcc.ed.ac.uk/∼ogsanet/

[26] The OGSI.NET Project. http://www.cs.virginia.edu/∼humphrey/GCG/ogsi.net.html

[27] Simple Object Access Protocol (SOAP) and XML Protocol (XMLP). http://www.w3.org/2000/xp/Group/

[28] Wason, J.L, Molinari, M, Jiao, Z and Cox, S.J. (2003) Delivering Data Management for Engineers on the Grid. Euro-Par 2003 Parallel

Processing,413-416

[29] Common Language Runtime (CLR) Overview. http://msdn.microsoft.com/library/en-us/cpguide/html/cpconCommonLanguageRuntimeOverview.asp

[30] Java Native Interface (JNI). http://java.sun.com/

[31] Microsoft Common Object Model. http://www.microsoft.com/Com/default.asp

[32] Java Virtual Machine (JVM). http://java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html

[33] Song, W, Keane, A.J, and Cox, S.J. CFD-Based Shape Optimisation with Grid-Enabled Design Search Toolkits. Proceedings of UK e-Science All Hands Meeting 2003, pp. 619-626.

[34] The Web Service Security (WS-Security) Specification. http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[35] Chappell, D. WS-SECURITY New Technologies Help You Make Your Web Services More Secure. MSDN Magazine, April 2003.

[36] PORT Mathematical Subroutine Library. http://www.netlib.org/port/

[37] Microsoft Internet Information Services (IIS). http://www.microsoft.com/WindowsServer2003/iis/default.mspx

[38] Microsoft ASP.NET. http://www.asp.net/Tutorials/quickstart.aspx

[39] Ewald, T. Programming with Web Services Enhancements 1.0 for Microsoft .NET. http://msdn.microsoft.com/webservices/building/wse/default.aspx?pull=/library/en-us/dnwse/html/progwse.asp

[40] Keane,A. J. OPTIONS Design Exploration System. http://www.soton.ac.uk/ ajk/options.ps

[41] Song,W, Keane, A.J, Eres, M.H, Pound, G.E, Cox, S.J. Two Dimensional Airfoil Optimisation using CFD in a Grid Computing Envirionment. Euro-Par 2003 Parallel Processing, Lecture Notes in Computer Science, 2790, 525-532

[42] ProEngineer. http://www.ptc.com, 2004

[43] Introduction to Fluent. http://www.fluent.com/software/fluent/

[44] Schwefel, H. Evolution and Optimum Seeking. Wiley, New York (1995).

[45] Keane,A. J. "Wing Optimization Using Design of Experiment, Response Surface, and Data Fusion Methods", J. Aircraft 40(4) pp. 741-750 (2003)

[46] Czajkowski, K., Ferguson, D.F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Vambenepe, W. The WS-Resource Framework. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf

[47] Czajkowski, K., Ferguson, D.F., Foster, I., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S. From Open Grid Services Infrastructure to WSResource Framework: Refactoring & Evolution. http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf