Integration of chargeable Web Services into Engineering Applications

Marc Molinari, Kushan Nammuni, and Simon J Cox

Southampton e-Science Centre

University of Southampton, SO17 1BJ

Abstract

We discuss an engineering case-study in which we address the issues of integration and secure consumption of a specialist software package for finite element meshing through a web service-based interface. This work focuses on demonstrating the end-to-end requirements of a framework allowing the transparent integration of the web service into the design workflow of an engineering user. After detailing an example engineering problem, we discuss server-side issues and required modules such as local accounting, security and resource management which provide a sample framework for a chargeable web service. We then demonstrate how the client can easily interface to and consume the web service from within the engineering scripting language Matlab.

1. Introduction

Numerical simulation and design optimisation processes in engineering applications frequently require specialist expensive software packages which often run only on dedicated hardware. Today's provision of such software is performed through ordinary licensing channels, often without the provision of hardware resources as part of the package. In the future, chargeable web services may offer opportunities to access hardware and software services over the internet via a variety of charging models [1,2].

Web services are based on open standards recommended by the W3C and thus provide flexible access methods which allow for uncomplicated consumption if appropriate technologies are used. It is essential to make these access tools easily accessible to the users and their preferred working environment.

We have developed the framework to integrate a chargeable finite element meshing facility as part of an engineering design calculation. A virtual "Grid Bank" provides the access to banking facilities both to the user and the service provider. The local accounting facility at provider level makes a pre-pay/pay-per-use scenario possible. The client to this infrastructure is an engineering user with access to Matlab as an engineering scripting language, consuming web service, Condor and Globus enabled resources.

In the following sections, we present a real-world sample engineering problem, detail the architecture and integration of the finite element meshing web service and discuss the consumption of this in a Matlab-based engineering workflow. Important aspects such as security issues and service provider accounting facilities are considered before we draw our conclusions and detail further work.

2. Engineering Application

As an example of a typical engineering application in the field of electromagnetic design optimisation, we have chosen a parametric design study of a component for next-generation integrated photonic devices [3]. Our approach, however, can be extended and applied in the same way to problems in other engineering and science-based application areas [4].

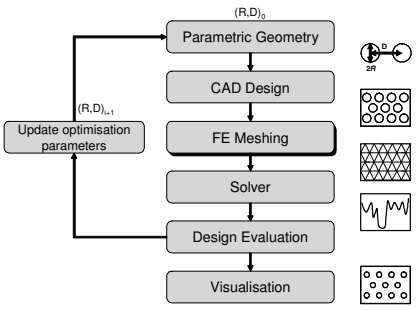

Figure 1: Typical workflow in engineering design optimisation, including call to a finite element meshing software package.

Suppose an engineer wants to optimise the electromagnetic transmission properties of an optical component of such a device, for example a Photonic Crystal (PC). A PC consists of a periodic geometric structure of holes etched into

a slab of dielectric material as shown in the right hand side column of Figure 1. The properties and density of the geometry determine the transmission properties of light through the crystal. By varying the radius parameter R and the distance parameter D, the transmission properties of the crystal will change until an optimised geometry set-up for the desired value of light transmission is achieved.

The optimisation of the design is an iterative process during which the geometric parameters are updated to give an improved design $(R,D)_{i+1}$. This process repeats itself until a satisfactory result has been achieved and can be visualized.

Figure 1 shows this typical design optimisation loop in an engineering workflow, containing the stages of design, mesh creation, solution and evaluation of the design.

Any one of these stages can be very complex and often requires engineers to buy and use commercial software packages running on specialized hardware or – the alternative we present in this paper – to integrate a web service which performs the desired task into the workflow, as indicated in Figure 2.

Here, we focus on Matlab as an engineering scripting language and detail in the following sections how such a service, providing a finite element meshing tool, can be integrated in a typical Matlab script by engineers without having to learn new programming languages or techniques.

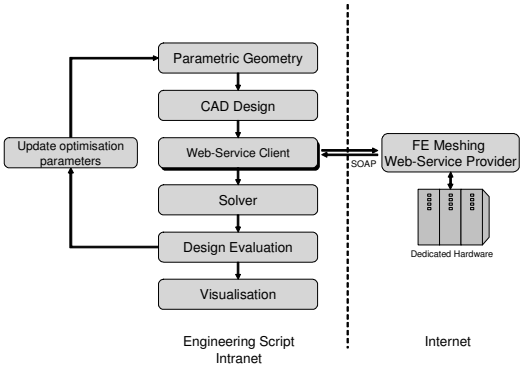

Figure 2: Workflow in engineering design optimisation, integrating a web service based finite element meshing application via the Internet through the SOAProtocol.

3. Chargeable FE Meshing Web Service

As a sample implementation of a functionality often required by engineering users, we demonstrate the implementation of a finite element meshing executable, FEMesh.exe, as a chargeable web service. This might in future be extended to chargeable Grid services which are discussed in [2].

We introduce an accounting module, resource usage reorder and a banking interface implementation to support chargeability using existing web service technology which is detailed in following sections.

The assumed underlying pricing model is based on a pay-per-use charging scenario. The business model is still in early stages, we assume a utility model or brokerage model.

Architecture

Our implementation represents a flexible, modular component based service oriented architecture in which we exploit existing web services technologies, which allows a conversion to future technologies using Grid Services

In this example, we have used a finite element meshing routine called FEMesh.exe – a package similar to the public domain tool Geompack++ [5] – as an exemplar FE meshing application. This is an executable which runs only on resources running the MS Windows OS.

Therefore, we have implemented a web services wrapper around the executable to make it accessible within the work flow design tool written as part of the GEM project [6].

A number of possible charging scenarios determine the accounting functionality required on the provider side as well as the interaction steps between consumer and provider. In this example, we employ the pay-per-use scenario where, users pay a flat rate per meshing calculation. Charging scenarios are depending on the problem domain, for example, an astronomical analysis scenario may differ from a mesh computation. The following list exposes some of the possible charging scenarios for the design optimisation problem domain:

- Perform the job on given time scale (i.e., within 1 hour);
- For the available funds (i.e., less priority, more time);
- Cost optimization (i.e., calculate the minimum cost);
- Time optimization (i.e., calculate the cost so that the job is performed in a minimum time scale);
- Specific execution parameters call for differing resource usage and cost;
- Guaranteed price (i.e., the SP specifies a fixed price for the service);

- Monthly price plan that allows a certain allowance and then pay per use;
- Number of iterations required for the job. (i.e. charge per iteration);
- Cost ceiling at which processing should be stopped;
- Advance reservation of resources at a negotiated price.

The main task of the service provider is to authenticate the user, offer a contract for use of the service, reserve required resources and charge accordingly for the requested job. A database (DB) stores user-related resource usage for the jobs (i.e., the total time, number of CPU cycles, etc.). The knowledge extracted from the DB and subsequently stored in a knowledge-base can at a later stage be used by the charging module to predict the charges intelligently for the above mentioned charging scenarios. The price calculator will compute the actual total charge for the task to be carried out. This will make use of the knowledge-base to calculate prices for the agreed charging model by inferring from existing data. In our particular implementation we use the IBM DB2 database [7] for storing this data as well as the knowledge base and for user credentials.

Usage of Web Services

The required steps to take before the web service can be used, can be summarised as follows:

(1) User obtains an account with Grid Bank via web page,
(2) User obtains an account with Service Provider,
(3) User transfers money from Grid Bank to Service Provider (pre-pay),
(4) User agrees charging mechanisms in a contract with Service Provider,
(5) User consumes Service using proxy certificate and agreed contract details and gets charged accordingly on the Provider's User account.

A number of possible charging scenarios determine the accounting functionality required on the provider side as well as the interaction steps between consumer and provider. In this example, we employ the pay-per-use scenario where users pay a flat rate per meshing calculation. Other charging scenarios are depending on the problem extent and factors such as time-scale of the job, resource usage, advanced reservation discounts, etc.

Web Service Framework

In figure 3, we can see the modules involved on the Service Provider site and note that the main tasks to be performed include user authentication, accounting, service provision, and resource management.

The main task of the service provider is to authenticate the user, offer a contract for use of the service, reserve required resources and charge accordingly for the requested job. A database (DB) stores user-related resource usage for the jobs (total time, CPU cycles, etc.).

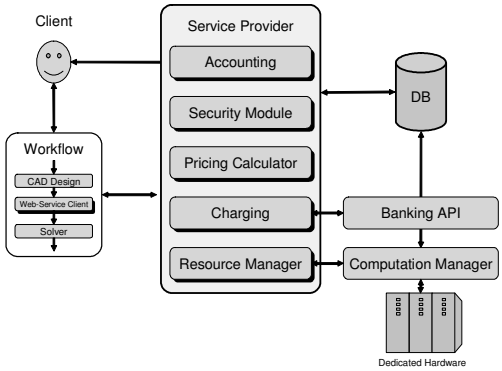

Figure 3: Web Service Provider framework showing required modules for accounting, user authentication and resource management. The database (DB) stores user account details as well as usage figures and service statistics.

The knowledge extracted from the DB can at a later stage be used by the charging module to predict/ modify the charges intelligently for the different charging scenarios. The price calculator will compute the actual total charge for the task to be carried out. The user is required to register with the service provider before consuming the service via the accounting module. This module provides a typical web-page based registration process where users can top-up their account prior to job submission. This reduces the complex implementation that would be necessary to manage chargeability issues on the server side.

Security Implementation

Security in business processes on the Internet is paramount. We base all our web communications between User and Grid Bank and between User and Service Provider on the Secure Socket Layer (SSL, [8]) and use proxy certificates based on the national UK e-Science PKI certificates [9] for authentication and authorization and for signing and encrypting SOAP messages which get passed between client and server. The user thus has to provide a

valid certificate which is signed by a Certificate Authority. We make use of the Java CoG kit [10] to validate the certificate chain to ensure the proxy certificate provided by the user was signed by the user's original certificate, which was signed by the designated CA.

4. Interfacing to the Web Service

To allow an engineer experienced with engineering scripting languages rather than web programming knowledge to transparently use the Web service in their engineering application, we have developed a tool which can be used within a graphical user interface based on the engineering software package Matlab. This interfaces directly to the underlying Java virtual machine and calls the available Web Service with the required parameters after a proxy certificate has been initialised and the surrounding workflow for consumption has been created. In a related tool, we have managed to create and consume Java proxy classes "on-the-fly" from WSDL documents provided by the Web service. With this package, the engineer can use the Web service from within their familiar environment in a transparent way which enables the efficient implementation of the finite element meshing functionality in their work.

As Matlab runs in a Java Virtual Machine (JVM) and thus provides a direct interface to Java classes and packages, a Java client to the web-service could be called directly from within the Matlab script. However, as we are proposing to re-use the service, we are going to request the WSDL document, convert it into Java code files using the Apache Axis wsdl2java package [11] and byte-compile the .java files into Java .class files. These can then be accessed more easily from within Matlab and the client files will exist on the local system even in the event of a loss of the JVM. The java classes then allow the consumption of the web-service from within a Matlab script.

As this conversion routine from WSDL to Java classes is very generic and allows for on-the-fly conversion of any WSDL document to a permanent client package, we have the added benefit of a relatively easy to integrate functionality for other web services.

In addition to the transparent integration, an engineer needs easy consumption methods. While it is possible to create Java classes from the WSDL document, the order of calling the class methods is not unique and immediately clear.

A first step to enable the workflow to execute is to set up a user account with the service provider. The provider creates an account and adds credit points to this account according to their payment policies (or free bonus points to attract new users).

The user will have a personal certificate from a third party and signed by a CA, or will obtain such an X509 certificate from the application provider. By extracting the signed message sent from the user to the service, the provider can uniquely identify the user.

This procedure of setting-up, checking and maintaining the local user account is best performed through a web interface which allows access from anywhere in the world where internet access is available.

```
...
% call CAD to mesh definition converter
mesh_def = cad_2_meshdef( cad );

% set web service parameters (shown is only a
% selection, this set-up has to be done only once,
% apart from the input file assignment)
WSdesc.name       = 'FEMesh-WS-0.1.1';
WSdesc.location   = 'https://ws-
  provider.org/xxx/FEMesh'
WSdesc.usercred   =
  'c:\temp\x509_user_cred_file.dat';
WSdesc.contractID =
  'c:\temp\x509_user_contractID.dat';
WSdesc.inputfiles = mesh_def.inputfiles;

[...]

% call FE meshing web service if available
if ws_isavailable( WSdesc )
  WSdesc.handle = ws_exec( WSdesc );
  mesher_result = ws_retrieve_results(
    WSdesc );
else
  error('Web Service not available...');
end

% call FE solver with file returned from WS
solver_opts.input_meshfile =
  mesher_result.filename{1};
result = FE_solve( solver_opts );
...
```

Figure 4: Matlab script implementing call to FE meshing web service. Lines starting with % are comments. The relevant parts of the script are in bold, the code at the beginning and end show how this integrates into the engineer's work-flow. The ws_exec and ws_retrieve_results functions hide the complexity of calling the web service and retrieving the results through Java interface functions from the user.

The actual consumption of the service is performed through a Matlab function wrapping

a small client script. The actions taken in this script are detailed in the sample shown in figure 4. This script has been auto-generated by the workflow tool and subsequently edited by the engineer to meet their specific requirements.

5. Conclusions & Further Work

We have demonstrated how to integrate functionality provided by a chargeable Web service into an engineering application.

The framework required includes a Grid Bank which may be replaced by an Internet based charging mechanism in the future. Local accounting, security and resource management modules at the service provider level are as important in the framework as a standardised but transparent and easy-to-use client software package for engineers.

At this stage, we have mainly focused on demonstrating the end-to-end requirements of our engineering example. Future work will use more generic underlying banking software (for example from the G-Markets project, [12]) and other licensing, charging and payment mechanisms along with use of 3rd party meshing tools at other sites (for example those from the University of Swansea).

More complexity can easily be added, such as complex electronic contract exchange, a number of business and pricing models and further extensions based on present and upcoming W3C recommendations and, for example, WSRF [13] or BPELWS[14].

Acknowledgements

We would like to thank the DTI for funding the GEM and G-Markets projects and acknowledge useful discussions with Steven Newhouse.

References

1. IEEE Computer – "Web Services", Oct. 2003, Vol. 36(10), pp. 35-62
2. S. Newhouse, J. Darlington, M. Asaria, A. Mayer, W. Lee, J. MacLaren, S. Cox, K. Nammuni, and K. Keahey. "Trading Grid Services Within the UK e-Science Grid", In: UK e-Science All Hands Meeting, p. 13-20, Nottingham, UK, Sep. 2003. ISBN: 1904425119
3. G Parker and M Charlton. "Photonic Crystals". IOP Physics World, Aug. 2000, Vol. 13(8), pp. 29-34
4. MH Eres, Pound GE, Z Jiao, JL Wason, F Xu, AJ Keane, and SJ Cox. "Implementation and utilisation of a Grid-enabled Problem Solving Environment in Matlab", accepted for publication in Future Generation Computer Systems, 2004
5. www.allstream.net/~bjoe/index.htm
6. GEM – Grid-enabled Electromagnetic Optimisation, http://www.soton.ac.uk/~gridem
7. RB Melnyk, and PC Zikopoulos. "DB2: The Complete Reference (Complete Reference Series)", 1st edn. McGraw-Hill Osborne Media (2001). ISBN: 0072133449
8. C Berg. "Designing Secure J2EE Applications and Web Services", 1st edn. Prentice Hall PTR, New Jersey (2003). ISBN: 0131402056.
9. P Kumar (ed). "J2EE Security for Servlets, EJB's and Web Services", 1st edn. Prentice Hall PTR, New Jersey (2003). ISBN: 0131402641.
10. Commodity Grid Kits. http://www.globus.org/cog
11. http://ws.apache.org/axis/
12. G-Markets, http://www.lesc.ic.ac.uk/markets
13. WS Resource Framework. http://www.globus.org/wsrf
14. BPEL for WS. http://www-106.ibm.com/developerworks/library/ws-bpel