

Making the Grid Pay - Economic Web Services

Jeremy Cohen, William Lee, Anthony Mayer, Steven Newhouse

London e-Science Centre, Imperial College London, South Kensington Campus,
London SW7 2AZ, UK
Email: lesc-staff@doc.ic.ac.uk

Abstract. The development of composable Web Services provides the technology that enables a true marketplace in computational services. Tradeable services, with charging and accounting, could realise hitherto unobtainable economic models of internet scale computing. We describe fundamental services that form the core of a future set of economic services, and how they operate to support computational markets. The services include a Chargeable Web Services, which implement the pricing port type, and Payment Provider services.

1 Introduction

The Grid presents a near utopian vision of a variety of high-quality resources available for immediate use by e-science and e-business users around the world. Such a vision may remain a distant goal but it is predicated on two important factors. The first is a distributed, scalable infrastructure, such as that provided by Web Services, to enable participants in the Grid to discover and use arbitrary remote services. The second is an infrastructure that allows service providers to be paid for providing services, thus allowing users to pay for their use as required in the same way as services are paid for in the ‘real’ world. Such a step is essential in providing a high-quality, well provisioned, services that can support both business and academic users. Without payment through a secure, trusted and transparent mechanism, Grid activity will remain confined to single organisations, or between actors who have a pre-established relationship.

This has been recognised within the UK’s Core e-science programme, supported by the Department of Trade and Industry (the government department with a responsibility for developing UK business) and has led to the formation of the ‘Markets for Computational Services’ project (<http://www.lesc.ic.ac.uk/markets>) to develop and explore the potential of such an infrastructure within the academic and commercial grid communities. Its participants include the regional e-science centres in London (lead site), the North West and Southampton, a variety of commercial partners including hardware vendors, application software vendors and service providers, and end users within the engineering and physics communities. The UK’s Grid Support Centre will deploy the infrastructure developed through the project throughout the UK e-Science Grid.

There are two main goals of the project: to develop an Web Service based infrastructure that supports the trading of services, and to explore a variety

of economic models within this infrastructure through its deployment across a testbed between the e- science centres involved in the project.

2 Architectural Overview

A web services infrastructure that enables payment for services is predicated on two distinct but related functionalities. Firstly, a mechanism must exist to enable prices to be set: two parties agreeing to trade some service must come to some mutual agreement or contract specifying the service and the price. Secondly, a mechanism must exist to ensure payment occurs as the contract is exercised.

We consider two web service port types to support these two economic aspects. By only using two port types, we maintain a simple model of interaction. A client interacts with a chargeable web service which supports the pricing port type (Section 3), and only with that service. In this fashion, the normal non-charging relationship between client and server is preserved - the only addition required to enable economic activity is the service provider implementing the pricing port type.

Monetary payment for the service is handled within a distinct unit, the Payment Provider Service. The chargeable service contacts the Payment Provider in isolation from the client, and as such the mechanism of payment is separated from the invocation of the service. This separation of concerns ensures the interaction between client and server is changed as little as possible from the non-chargeable interaction, guaranteeing encapsulation and clean interfaces. We discuss the interactions and interfaces of the two web service port types separately.

3 Pricing Port Type

3.1 Pricing Port Type Architecture

The *Pricing Port Type* is supported by any web service which wishes to charge directly for its services. As such any web service which implements this type is known as a *Chargeable Service*.

The key feature of the pricing port type is the negotiation process, by which the client and the service come to agree on a contract (with a price) for the invocation of the service. The process is iterative with party and counter-party (client and service) exchanging a document, making changes on each iteration to reflect their current bargaining position.

The document structure is given by the WS-Agreement [1] specification. The contract is a set of negotiation terms, which may be generic or domain or service specific. The iteration process continues until the terms describing the nature of the service provided, and in particular the price charged, are settled upon by both parties.

A client who wishes to use a chargeable service does so according to the communication pattern illustrated in Figure 1.

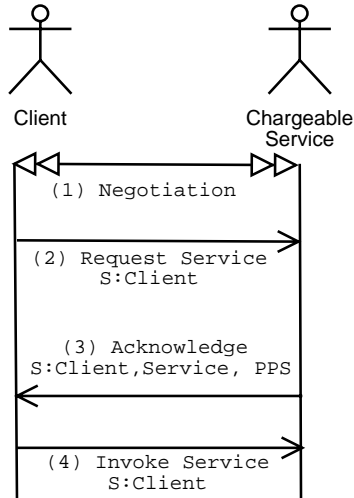


Fig. 1. Chargeable Service Activity Diagram

3.2 Pricing Port Type Interface

The following operations are supported:

AgreePrice This operation supports the negotiation of the service contract. The input is a document representing the counter-party’s proposed contract, as specified using the WS-Agreement terms. The output is the response - a corresponding set of agreement terms that the counterparty may choose to accept or continue to negotiate.

Pay Once the contract has been settled using the AgreePrice operation, the client (user) of the service can invoke the pay operation. This requires passing the chargeable service details of the client’s Payment Provider Service (as detailed below).

Usage This is not a separate operation of the interface, rather a set of meta-data carried within the *context* of the conventional invocation of the service’s ports. This contextual information includes a signed document that provides a unique reference for the invocation, which is then used to account for the usage and to charge accordingly.

The implementation of the ‘AgreePrice’ port depends upon the negotiation logic (and hence the usage policy) of the chargeable service, and is therefore not common to all services.

4 Payment Provider Service

4.1 Payment Port Type Architecture

The *Payment Provider Service* port type is a common interface to support the transfer of money via an underlying, ‘real world’ payment mechanism. There are many such payment systems in use today, and a generic Grid model must be agnostic as to the choice of underlying payment systems. Instead, by providing a standard Web Services interface to such models, we move towards a goal of an open market in services.

From this we can see the requirements of a payment provider service:

1. The port type must be abstract, so as to encompass a variety of different real payment mechanisms.
2. It must support autonomous delegated payment authorisation
3. It must support existing, commodity security models available to Web Services technology and used within the Grid Community i.e. X.509 certificates.
4. The payment model must resist replay attacks.

The participants within the model are:

- The payer (client), who is paying to use a service, and provides the authorisation for the payment to proceed.
- The payee (service provider), who is being paid for access to a web service.
- The Payment Provider Service, that provides an abstraction to the underlying monetary transaction between the payer and the payee.

We note that the client participant need not be the ultimate recipient of the service - the client can be an autonomous agent or broker, as long as the client possess the requisite authority to sign certificates on behalf of the ultimate receiver of the service. This can be achieved through short lived proxy certificates, which allow the proximate authority to impersonate the ultimate one, but bound security concerns in time.

The communications between the service provider, the payment provider, and the client (whether they be the payer, or a proxy authority), are illustrated in Figure 2. The dotted lines indicate the client’s interaction with the chargeable service, which is opaque to the Payment Provider Service.

This communication pattern is as follows:

1. The client (who has a relationship with a Payment Provider Service), has agreed a price for a service with a Service Provider. She sends a signed message giving the details of her account with the Payment Provider’s contact details.
2. The service contacts the Payment Provider Service, with a signed document that contains the previous communication from the user, requesting that a payment to the service be authorised.

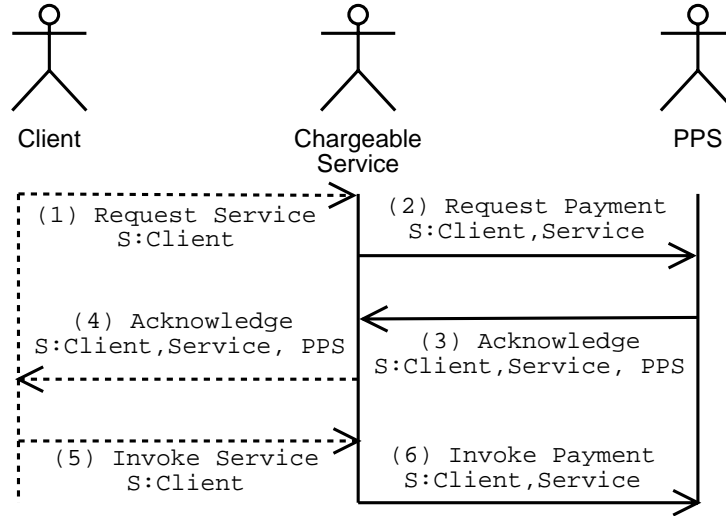


Fig. 2. Payment Provider Activity Diagram

3. The service will check the credit of the payer, and reply to the service with a signed document, indicating that the payment and account is valid. In this way the server can offer a service in the knowledge that the payment will be honoured by the Payment Provider.
4. The service then contacts the user (again with a signed document), to ensure that the user is aware of the agreement. The user thus has a record of the transaction details.
5. The user invokes the service, passing a signed document with a unique reference for the invocation.
6. Upon performing the service, the payee contacts the Payment Provider, passing the signed reference. The payment provider then performs the payment for the service as previously agreed.

It can be seen that this model prevents replay attacks, while at the same time allowing repeat invocations of the service, with reciprocal payment. A replay attack would have the service repeating step 5 multiple times, each time collecting money authorised by the client in step 1. The Payment Provider Service can detect this forgery, as the same signed reference number will be passed for each transaction. At most one such transaction will therefore take place.

Conversely, if the contract agreed is a Pay-Per-Use one, with each invocation of the service requiring a micropayment, there is no need to complete the entire authorisation cycle each time. Instead the user makes repeated step 5 communications, each with a distinct reference, and the service simply passes these to the Payment Provider.

It should be noted that the means by which the Payment Provider performs the monetary transaction is not included in the protocol. This is beyond the scope of the specification, and is essentially implementation dependent. It is clear that this mechanism is abstract enough to encompass multiple implementation types, and by using conventional security mechanisms (signed documents) can resist replay attacks.

4.2 Multiple Payment Provider Services

In many cases the service provider may be an intermediary service, which itself possesses a relationship with an existing Payment Provider Service. Though this extends the client-server-payment provider model, it simplifies the reconciliation of payment upon transaction, as the two parties (client and server) are mirrored by their respective payment providers, who negotiate over the transfer of money using a single interface.

The extended activity diagram is shown in Figure 3, in which the server's Payment Provider is identified as 'PPS-SP', and the original client side Payment Provider is identified as 'PPS-U'.

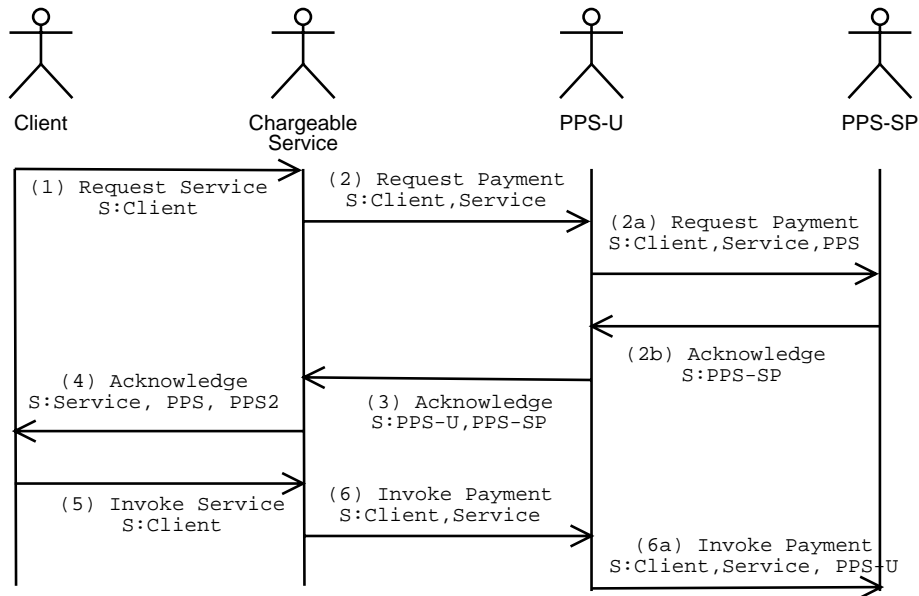


Fig. 3. Multiple Payment Provider Activity Diagram

The modifications to the communications are as follows:

2. The service contacts the client's Payment Provider, passing details of its own Payment Provider.

- 2(a). The client's Payment Provider contacts the server's Payment Provider, to establish a relationship in which the transaction can occur (passing account details and the like).
- 2(b). The server's Payment Provider acknowledges 2(a).
- 6(a). The client's Payment Provider contacts the server's Payment Provider to perform the monetary transfer.

4.3 Payment Provider Interface

The payment provider interface consists of the following operations:

PaymentSystem Returns a document describing the characteristics of the underlying payment system.

VerifyUserTransaction This determines whether the details provided by the client to the service provider are correct, and that the transaction can take place. This operation accepts a transaction, and returns an authorisation for that transaction. In effect, a credit check on the transaction takes place. Input parameters enable this operation to put a hold on the resources needed to pay for the transaction. This operation is used in Step 3 of the communication.

CompleteTransaction Accepts a verified transaction document, signed with a unique reference (Step 6 of the exchange above).

CompleteUserTransaction Accepts a document from another Payment Provider, indicating the monetary exchange is to take place (Step 6(a) of the multiple PPS model).

5 Conclusion

The prototyping of these two port types provides the initial groundwork for higher level economic services, in particular intermediaries that buy and sell contracts on services. By using conventional web service technology it is possible to provide the foundation for a true web based market in computational services.

References

1. Grid Resource Allocation Agreement Protocol. <https://forge.gridforum.org/projects/graap-wg>.