GWD-R (draft-ggf-gesa-services-1)
Grid Economic Services Architecture (GESA)
http://www.gridforum.org/3_SRM/gesa.htm

Editors:
S. Newhouse, LeSC

June, 2003

Grid Economic Services

Status of this Memo

This document provides information to the community regarding the specification of the services within the Grid Economic Services Architecture (GESA) being developed within the GESA-WG. Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

Abstract

The Open Grid Services Architecture (OGSA) provides an infrastructure for virtualising resources of many types (compute, storage, software, networking etc.) as Grid Services. The infrastructure for building these basic grid services is being defined elsewhere within the Global Grid Forum. Although mechanisms will exist for defining these services it is unlikely that any sustainable infrastructure will be provided by any non-research organization without financial compensation. For Grid Services to be provided on demand (i.e. to provide the utility infrastructure that has always been the vision of the Grid) organizations will want to be paid for providing these resources.

The purpose of this document is therefore to define the additional service data and ports needed to describe the economic grid services – the enabling infrastructure – rather than to describe the economic models that will be built on such an infrastructure.

GLOBAL GRID FORUM

office@gridforum.org
www.ggf.org

Full Copyright Notice

Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

Explicit statements about IPR should not be included in the document, because including a specific claim implies that the claim is valid and that the listed claims are exhaustive. Once a document has been published as a final GFD there is no mechanism to effectively update the IPR information. Authors should instead provide the GGF secretariat with any explicit statements or potentially relevant claims.

Contents

1 Introduction

The ability to virtualise any resource as a service through a standard framework, such the Open Grid Services Architecture (OGSA), will enable many different forms of interaction between these diverse service offerings. However, the provisioning of these services is currently dependent on 'best effort' from the academic and research community. For Grid Services compatible with the OGSA to be provided reliably to the users in a community then the users must expect to fund these services in some manner. By integrating the ability to charge for Grid Services within the core OGSA infrastructure we expect to enable new models of service provisioning such as utility computing.

One such effort to develop such an architecture is taking place in the UK through the successful funding of a UK e-Science Core programme project – A Market for Computational Services. As part of this activity we are developing an infrastructure to enable the trading of Grid Services as defined through the OGSA. This document will define extensions to the standard Grid Services that will enable the construction of such a marketplace. By definition any such marketplace must support interoperable protocols and we welcome contributions from other organisations working in this area to build a cross-community infrastructure.

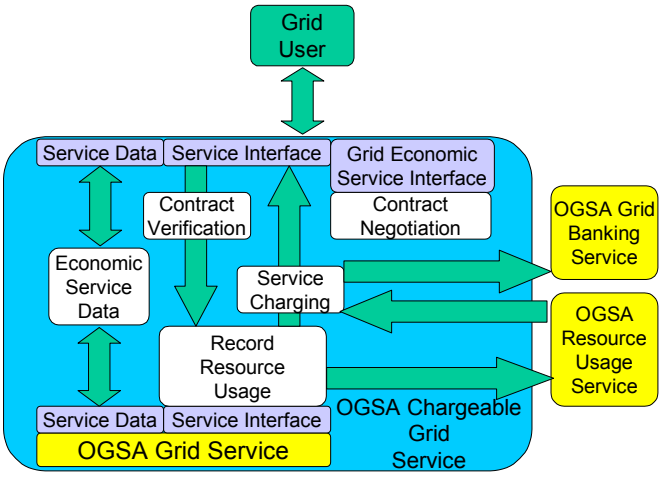
We expect the standardisation of this infrastructure to be a community effort taking place with the Grid Economic Services Architecture (**GESA**) Working Group within the Global Grid Forum (**GGF**). We expect these efforts to interact with other activities within the GGF, notably the Resource Usage Service (**RUS**), the Grid Resource Allocation Agreement Protocol (**GRAAP**), the Open Grid Services Architecture (**OGSA**), Open Grid Services Infrastructure (**OGSI**) and the Usage Record (**UR**) Working Groups.

2 Overview

2.1 Architecture

We believe the mechanisms needed to trade services are well established from work in traditional economic areas. We therefore intend to deliberately exclude the detailed mechanisms as to how we will price these services from this document other than for illustrative purposes. We will define how these different pricing methods may be integrated into GESA. Likewise the mechanisms to choose one service over another are also beyond the scope of this initial document. However, we will focus on the static and dynamic meta-data that needs to be generated and maintained within the Service Data Elements (**SDE**) exposed through the relevant ports defined by the Open Grid Services Infrastructure (**OGSI**).

A strawman architecture of such an infrastructure is described below showing how the Computational Grid Service (CGS) which wraps the Grid Service that is to be sold, interacts with the Grid Banking Service (GBS) and the Resource Usage Service (RUS).

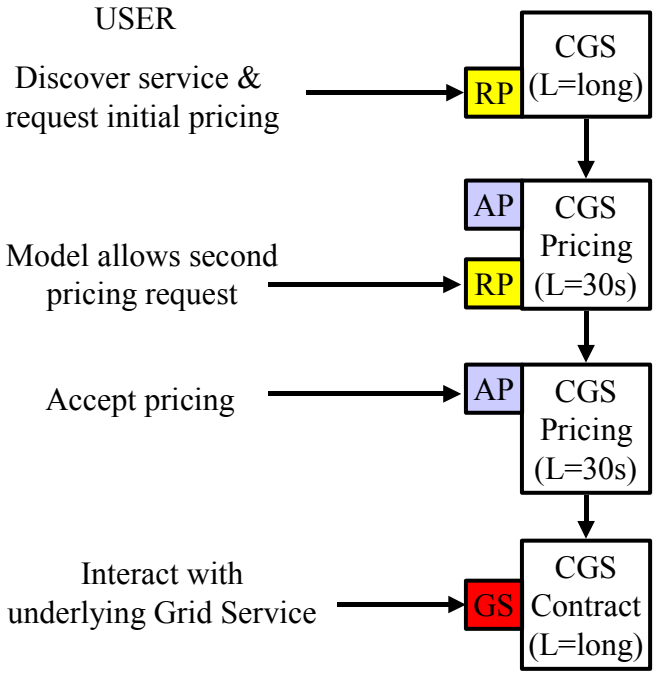


Early discussions with the GGF identified one key requirement: that the underlying OGSA service interface should not be changed, only extended, by the wrapping of a Grid Service as a CGS. This would allow existing clients to interact with a CGS even if the client interface had been generated for the underlying Grid Service.

This document defines the service data elements and service interface for the CGS in section 3 and the Grid Banking Service (GBS) in section 4. A variant of the GBS, the GBSHold Service which allows ‘reservations’ to be made on a user’s money is defined in section 5. Section 6 contains a simple example as to how a system using these protocols might work. This requires the use of the Resource Usage Service (RUS) which is defined in a similar document from the Resource Usage Working Group.

This basic architecture exploits the transient nature of a Grid Service to encapsulate the cost of using the service within its SDE’s. All changes in state of the Grid Service

(from the initial advertisement, establishing the cost of its use, to the acceptance of this cost, through to its eventual use) are encapsulated through the creation of new services.



This sequence shows how the user finds a service and requests a price through the RequestPricing port. The pricing is encapsulated in a short lived service (30s in this example) which is not acceptable to the user and the chosen economic model supports a second pricing round pricing request which is triggered by the second call to the RequetPricing port to produce a second short-lived service. The user has only two choices: to reject the price and let the service destroy itself after 30s or to accept the pricing (acceptPricing port) which produces a long-lived service specifically created for the user. The pricing of this service may take two stages (as in this example) a single stage, or many stages. The detailed protocols need to support this form of interaction is described in this document.

2.2 Definitions

Throughout this document we will use the term ‘user’ as a generic term for a client to a CGS which may be an interactive user client, a broker acting on the user’s behalf or any other such entity.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2.3 Scope

In the remainder of this document we define the structure of the CGS and GBS, and how they interact with other services such as RUS. For each of these services we define the:

- Service Data Elements: The additional SDE’s that are needed within the Grid Service to express the service’s economic meta-data.

- Service Interface Definition: The operations needed to support interaction with the Grid Service.
- Implementation: Observations on implementing the operations
- Other Issues: Our goal is to build on the OGSI and related standards. In some areas these may need clarification or to be developed further.

We also define subsidiary services that are needed to support these primary services.

3 The Chargeable Grid Service (CGS)

The CGS represents the abstraction of a Grid Service that has been enabled to support economic interaction.

3.1 Service Data Elements

The SDE's provided by the CGS are in addition to those defined within the GSS. These contain static and dynamic meta-data relating to the economic use of the service and are contained in a new SDE hierarchy.

SDE	Occurrence	Provided By…	Comment
Pricing	1+	Service Admin	Supported pricing mechanisms
Currency	1 + Static	Service Admin	A declaration of the currencies provided by a GSH that are acceptable.
Usage	0/1 Static	Service Admin	A GSH to a trusted RUS.
Price	1	Service Admin	Price generated through the pricing mechanism.
Resources	0+	Service Admin	The consumed resources that will incur cost to the user.
Liability	0/1 Static	Service Admin	Route to 'human compensation'.
Compensation	0 + Static	Service Admin	Pays a proportion of the service cost to the user if the service fails to deliver.
Refund	0 + Static	Service Admin	Refunds a proportion of any paid charges to the user if the service fails to deliver.
Testimonial	0 + Dynamic	Third Party Service	A digitally signed declaration from a Grid entity as to the reliability of the CGS

All economic SDE's are contained with a SDE's of type:

```
<serviceData name="gesa:economicSDE">
   …
</serviceData>
```

This list of SDE's is not exhaustive and should be expanded and adapted as the requirements from the economic models develop. For instance, instead of using real currency within the refund or compensation SDE's a service may like to give credit. This could be represented as a currency exchangeable only with the services run by a specific service provider. The SDE's constitute a service specific advertising element and some of these SDE's may only be relevant at different stages of the CGS lifetime.

3.1.1 Pricing

The primary purpose of a CGS is to let other Grid entities know how to obtain pricing information relating to the use of the service.

```
<gesa:pricing name="FixedPrice" >
  <gesa:Duration default="30" maximum="60" />
</gesa:pricing>
```

A CGS MUST support a pricing SDE for each pricing mechanism supported by the service and these are differentiated by their different names and the service data elements that they encompass. There MAY NOT be any difference between a pricing mechanism named "DutchAuction" and one named "FixedPrice". The service provider may add attributes to the pricing service data to describe the mechanisms used within the pricing mechanism. It is therefore possible to have a pricing system named "FixedPrice" and "FixedPriceWithCompensation" which have different pricing strategies as they offer different levels of compensation – some and none.

The presence of a `pricing` element implies that there is a pricing capability accessible through the `CGS::requestPricing` port that will, if invoked, produce a new service instance containing pricing information within a `price` element. Within the `pricing` element we define the default and maximum lifetime of any quotation (i.e. the underlying service) provided by the CGS.

3.1.2 Pricing Attributes

Within the pricing element the service provider MUST provide elements that characterise the pricing method.

```
<gesa:pricing name="FixedPrice" >
  <gesa:FixedPrice />
</gesa:pricing>
```

The above example provides an example of how to describe a CGS's pricing mechanism by using a fixed pricing mechanism. The elements used to classify the pricing mechanism (i.e. a very simple lightweight ontology) are given below:

Element	Definition
FixedPrice	The price for the service is set in a single non-negotiable stage.
Auction	Indicates that the price is set through a multi-stage action.
EnglishAuction	Indicates a particular approach to setting a service price.

[SJN: Further elements (obviously) need to be defined here.]

3.1.3 Currency

All transaction within a CGS will incur some 'cost' on a GBS for the resources that are consumed. This cost may be charged in real money (through some later off-line reconciliation) or through some form of site/organisation specific service tokens.

```
<gesa:pricing name="FixedPrice" >
…
  <gesa:currency name="HeyPounds" email="cash@hey.ac.uk" />
</gesa:pricing>
```

The currencies that are usable within each CGS are declared through one or more of the above elements. In order to complete a transaction the service must 'know of' a GBS that supports this currency type and the grid entity requesting to use the service. A GBS that supports this currency may be found by searching the service registry and it's interface and SDEs are defined later.

3.1.4 Usage

All invocations within a service are recorded in a Resource Usage Service instance. This service is used to collect the resources consumed by a service for the purposes of calculating a charge to the user for service use.

```
<gesa:resourceUsage GSH="GSH for the RUS" />
```

3.1.5 Price

Once the price has been generated by invoking the `CGS::requestPricing` port it needs to be displayed through a SDE within the new service instance. This allows the user to hold several service instances that MAY be used by the user and to examine the economic state of each.

```
<gesa:priceRate currency="HeyPounds" timeperiod="60" />
  <ur:CPUDuration>1</ur:CPUDuration>
  <ur:memory units="GB">10</ur:memory>
</gesa:priceRate>
<gesa:priceMaximum currency="HeyPounds" />
  <ur:processors>100</ur:processors>
</gesa:priceMaximum>
<gesa:priceMinimum currency="HeyPounds" />
…
</gesa:priceMinimum>
<gesa:priceTotal currency="HeyPounds" />
…
</gesa:priceTotal>
```

The price may be set for a particular resource by different measures. One approach is to charge the consumption of resources at a rate (e.g. Mb/s), total consumed resources (e.g. number of processors), maximum value (e.g. temporary disk space), etc. This example indicates that the cost for using the CPU will be charged at 1 HeyPounds per unit time and memory at 10 HeyPounds per GB of memory used per unit time. The maximum number of processors used by the user will be charged at 100 HeyPounds for each processor used. The default time period is 1s in this example the costs are given per 60s.

3.1.6 Consumed Resources

Any service invocation will consume a wide range of resources. However, a service provider may only be interested in a relatively small subset of these resources for the purposes of deciding a price to use a service. The Usage Records Working Group within the GGF have defined an initial subset of base properties, such as:

- Network
- Disc
- Memory
- Wall Clock Time
- CPU Time
- Node Count
- Processors

which could be used as part of the service pricing policy. The resources that the CGS will charge for are specified in the `gesa:pricing` element. This element also specified if an estimate if this resource is required by the service to provide ANY pricing for service use. The default value for the 'estimateRequried' attribute is false.

```
<gesa:pricing name="FixedPrice" >
…
  <gesa:chargedResources >
    <ur:memory />
  </gesa:chargedResources>
  <gesa:chargedResources estimateRequired="true" >
    <ur:cpuTime />
    <ur:processors />
  </gesa:chargedResources>
</gesa:pricing>
```

3.1.7 Liability

Defines the organisation responsible for providing the service. This is an area in need of further exploration. Effectively this is an informational SDE element which a user agent may search for (or not) and provides the required information if needed by the user.

```
<gesa:economicSDE>
…
<gesa:liability organisationName="London e-Science Centre"
                              email="lesc-admin@doc.ic.ac.uk" >
    Complaints Department
    London e-Science Centre
    180 Queen's Gate
    London, SW7 2AZ, UK
</gesa:liability>
…
</gesa:economicSDE>
```

NB: There should probably be an additional element in here with a certificate & digital signature allowing this statement of liability to be authenticated in an automatic manner.

3.1.8 Compensation

A statement of compensation is required for any organisation offering a service for monetary reward. The level and complexity of compensation MAY vary from one organisation to another and from different pricing methods offered by the same organisation or even the same service. For instance, "GoldStarFixedPrice" might provide some specified compensation while. We consider here some simple cases:

```
<gesa:pricing name="FixedPrice" >
…
  <gesa:compensation percentage="0" />
</gesa:pricing>
```

This option allows the client is to refund to the client the agreed cost of invoking the service even if the server defaults on the delivery of the service before it is invoked. This element will, by default have the amount contained within the 'percentage' attribute refunded to the client on failure:

- `percentage="0"` : means that the client will receive no compensation for any service failure.
- `percentage="100"` : means the client will receive in compensation the agreed cost of using the service even if the client has not yet paid for any part of the service.

- `percentage="200"` : means the client will receive in compensation twice the agreed cost of using the service.

By default the percentage value is set to zero meaning the client will receive no compensation for any service failure. Any non-zero positive value of this variable will result in the service provider paying out to the client with no income if the service fails.
[SJN:How do we handle (if we need to?) staged payments: 10% on reservation, 60% on job startup, 30% on completion for a job.]

3.1.9 Refund

```
<gesa:pricing name="FixedPrice" >
…
  <gesa:compensationRefund percentage="100" />
</gesa:pricing>
```

This option allows the client to obtain a full refund of any money paid to the service provider if they do not deliver on the service. This element will have the amount contained within the 'percentage' attribute refunded to the client on failure:

- `percentage="0"` : means that the client will receive no refund for any service failure.
- `percentage="100"` : means the client will receive a full refund for any service failure, i.e. if money is deducted from the client's account it will be refunded.
- `percentage="200"` : means the client will receive a full refund in addition to an equal amount of compensation of the money deducted.

By default the percentage value is set to 100 meaning that if the service fails to deliver then, from a financial perspective, all cost transactions are rolled back.

3.1.10 Testimonial

Many usage scenarios include some mechanism to 'rate' the 'quality' of a service. This is from a technical context a fairly ill-defined problem that could be resolved in several ways:

- Broker: The broker collects services instances, testing them and adding rating information into the meta-data before repackaging the service as one that they can provide.
- Dynamic SDE: An alternative approach is to provide clients with the opportunity to update the SDE with their views as to the service's performance. This could include text, a numerical rating and a digital signature to provide credibility.
- Testimonial Server: Independent third party service that maintains a list of services and user supplied comments (positive/negative).

Testimonial elements have not currently been defined.

3.1.11 Product

Although it is possible to encapsulate the 'product' that is being sold within the Grid Service itself, this does not always fully capture the behaviour of the service being sold. For instance, the Grid Service may sell access to a mechanism to download a product, e.g. an operation that downloads an MP3 track, retrieve an electronic book, etc. One alternative to this approach is to encapsulate each product and pricing mechanism within a separate Grid Service.

```
<serviceData name="gesa:economicSDE">
  <gesa:pricing name="FixedPrice" >
    <gesa:Duration default="3600" maximum="3600" />
    <gesa:chargedResources>
      <ur:invocation/>
    </gesa:chargedResources>
    <gesa:product element=http://softwareprovider.com/schema.xml>
       <sp:availablePlatforms name="redhat-8.0" />
       <sp:product name="SicLib" version="1.2"/>
       <sp:duration time="24h" />
    </gesa:product />
    <gesa:product element=http://softwareprovider.com/schema.xml>
       <sp:availablePlatforms name="solaris-2.8" />
       <sp:product name="SicLib" version="1.2"/>
       <sp:duration time="24h" />
    </gesa:product />
  </gesa:pricing>
  <gesa:currency name="SciPounds" email=cash@softwareprovider.com />
     …
</serviceData>
```

3.1.12 Unresolved Issues

Items still to be resolved:

- The Compensation and Refund elements are activated on the 'failure' of the service to deliver on something (e.g. SLA). How is this failure detected? Through resource monitoring provided by RUS?
- Resource specific compensation mechanisms? Can resources be refunded?
- Is an insurance or warranty action needed beyond the compensation mechanism? Is this an extension of the testimonial action?
- How is this linked into Service Level Agreements (SLA)?

3.2 Service Interface Definition

We propose that the Grid Economic Services Interface (GESI) contain a number of ports to facilitate the GESA. The first of these is a Factory port to allow the creation of new instances of this particular CGS. It is envisaged that many of these CGS will have a multi-stage process to defining the final cost of the service to the user, e.g. negotiation, auctioning etc. To enable each mode of interaction the initial act of any CGS on being contacted by a user will be to create a new service instance to deal with the requested interaction method.

3.2.1 CGS::requestPricing

The `requestPricing` is a service port provided by the GESI that will create a new service instance containing information relating to the price charged for using the service. This port extends the `Factory::createService` port type. A service containing this port has the ability to provide a quotation for the use of the service. This quotation is encapsulated within a new service instance created by this port.

Input

- *TerminationTime* (optional): The earliest initial termination time that is acceptable to the client. This is effectively the length of time that the client wishes to retain the right to use the service. If not specified then it defaults to the duration in seconds specified by the `gesa:defaultDuration` SDE. After this time the service instance, and therefore the right to use the service, will be destroyed through the lifetime management provided by the container.

- *EconomicParameters* (optional): This factory-specific element contains data used by the GESA Factory element to help instantiate a new service instance and set the cost for its use. The EconomicParameters element MUST contain the following child elements:
    - *PricingMechanism*: This element specifies the pricing mechanism that is to be used.
    - *Product (Optional/Required)*: If multiple product elements are specified within the economic SDE then one of these must be specified within the pricing request.
    - *AllowedUser*: This element specifies the distinguished name of the users (or other user agents, e.g. brokers) that are allowed to access the created service. Multiple elements are allowed.
    - *Currency*: Specifies the currency that will be used to record payment. If only one currency element is specified by the CGS this element becomes optional. If multiple currency elements are specified in the SDE this becomes required.
    - *ConsumedResources*: Elements specifying the estimated consumed resources that the service invocation will use. This may be used by the CGS to adjust the returned costing.
    - *ServiceTerminationTime*: The time beyond which the user will no-longer require the service if they decide to make use of the service offering. This element may need to include 'notBefore' and 'notAfter' to define an advanced reservation. [SJN: Obvious links to GRAAP & SLA's here!]

Output & Faults
As defined in the OGSI specification document.

The pricing for the service use is encapsulated in the new service instance within the `price` element. The only mechanism for changing the price of a service is through the creation of a new service instance within the `requestPricing` element. Therefore at any point in this process a GSH with a `price` element within the SDE encapsulates the cost of using this service instance while a `pricing` element represents the presence of a `requestPricing` port and a means of creating a new price. This approach allows the 'root' service instance to declare how prices to be set but to make the final price a result of a multi-stage negotiation through a series of service instances.

Any use of the underlying CGS interface MUST result in failure until the price for using the service is explicitly accepted by invoking `CGS::acceptPricing`.

[SJN: It is not inconceivable that some services may wish to charge for providing a quotation. Could this be defined by placing price elements in the initial 'root' service?]

3.2.2 CGS::acceptPricing

This port MAY only appear if a `price` element is contained within the SDE. By invoking this port, which extends the `Factory::createService` port type, a new service instance is created embedding the terms and conditions in the new service. The existing service is destroyed as the quotation it refers to is no longer valid

as it has been accepted by a user. The new service provides access to the underlying Grid Service to the specified user community and will record the service invocations in RUS and calculates the resulting cost for recording in the GBS. This instance of the CGS, from a user perspective, is identical to the Grid Service it encapsulates. The overall lifetime of this new service is set from the parameters used by the proceeding `CGS::requestPricing` call. Within those constraints it MUST support standard OGSI lifetime management interfaces to manage the service lifetime.

Input

- *None*

Output

- *Contract*: Returns a signed XML document consisting of the economic SDE's signed by the service provider (i.e. the hosting environment or host certificate). This provides the user with a document (for offline storage) stating the terms and conditions for using the Grid Service which cannot be denied by the service provider at a later date.

Faults

As defined in the OGSI specification document.

3.3 Other Issues

The new service needs to retain a reference to the GSH that it was created from to enable coordination between different service instances. Is this already in the GSS? Should there be a mechanism to extend the lifetime of a quotation?

4 The Grid Banking Service (GBS)

The GBS provides a service to a payment infrastructure that is itself defined outside this document. Our scope within this section is to define the interaction between the GBS and other entities within the GESA (e.g. the CGS, a user). No implementation details are specified within this document, however the GBS could be implemented by any infrastructure with an account based abstraction. This could include systems based around electronic cash, credit cards, accountancy packages with periodic reconciliation, pre-paid accounts, service tokens, etc.

The 'currency' used in these transactions need not be recognised or supported by a large community. A currency could relate to service tokens allocated within a specific service centre or virtual organisation. No discussion is made here of converting between currencies but such functionality is easily envisaged. If the CGS is willing to accept more than one currency to pay for service usage then this may be specified within its economic SDEs.

4.1 Service Data Elements

4.1.1 Currency

The currency is the greatest differentiator between different instances of a GBS. The "`gesa::currency`" element declared earlier should be used to indicate the currency supported by this GBS.

```
<gesa:currency name="HeyPounds" email="cash@hey.ac.uk" />
<gesa:backer organisationName="London e-Science Centre"
                              email=lesc-admin@doc.ic.ac.uk >
    Banking Department
    London e-Science Centre
    180 Queen's Gate
    London, SW7 2AZ, UK
</gesa:backer>
```

NB: There should probably be an additional element in here with a certificate & digital signature allowing this statement of liability to be authenticated in an automatic manner.

4.1.2 TrustedUser

```
<gesa:TrustedUser
  name="/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=steven newhouse" />
```

Element that places the specified user into the role of a trusted user.

4.1.3 PrivilegedUser

```
<gesa:PrivilegedUser
  name="/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=steven newhouse" />
```

Element that places the specified user into the role of a privileged user.

4.2 Interface Definition

The GBS has to support the following operations within a GridBanking port type. As the current authorisation model for GridServices has yet to be defined we use the following classification for these operations:

- Un-privileged: A normal GSI authenticated client connection is sufficient.
- Trusted: A GSI authenticated client whose DN is registered as an account holder in the GBS or is contained in the TrustedUser SDE (defined earlier).
- Privileged: A GSI authenticated client whose DN is contained in the PrivilegedUser SDE (defined earlier.)

In the above and the following DN is defined as the Distinguished Name of the X.509 certificate.

4.2.1 GBS::isDNAccountHolder

This operation determines if the specified DN has an account with this GBS instance. This is used by CGS and other entities to validate that the user exists.

Input

- *DN*: String parameter containing the DN of the user. The presence of a user's account in a GBS should not be public information [SJN: Or should it be advertised in the GBS SDE?].

Output

- *Result*: Returns true (the account exists) or false (the account does not exist).

Fault

- *Fault*: Any fault that occurred.

This is operation is available to trusted clients, i.e. a client of a bank is allowed to query the existence of other clients.

4.2.2 GBS::creditCheck

Before agreeing to provide a service to a client it may be necessary to check that the client has the funds available to support the proposed cost of the service invocation.

Input

- *DN*: String parameter containing the DN of the user.
- *Amount*: The amount of the currency that the client is asking to be available.

Output

- *FundsAvailable*: Returns true if the funds are available and false if they are not.

Faults

- *NoDNFault*: The user's DN does not exist in the GBS.
- *Fault*: Any other fault.

This operation is only available to trusted clients.

[SJN: Should this be in the SDE – I'd say not as it is 'private' data and not sure the state of SDE ACL's]

4.2.3 GBS::getLastTransactions

Allows a user to retrieve their recent transactions or a privileged user to view another user's recent transactions.

Input

- *DN*: String parameter containing the DN of the user.
- *NumberOfTransactions*: Specify the number of transactions that should be returned.

Output

- *Statement*: An XML document containing the details of each transaction.

Faults

- *NoDNFault*: The user's DN does not exist in the GBS.
- *Fault*: Any other fault.

4.2.4 GBS::getTransactionsByDate

Allows a user to retrieve their recent transactions or a privileged user to view another user's recent transactions by specifying a date range.

Input

- *DN*: String parameter containing the DN of the user.
- *StartDate*: Specify the date from which transactions should be viewed.
- *EndDate* (Optional): Specify the date before which transactions should be viewed. If not specified this is taken to be the current date.

Output

- *Statement*: An XML document containing the details of each transaction.

Faults

- *NoDNFault*: The user's DN does not exist in the GBS.
- *Fault*: Any other fault.

4.2.5 GBS::transferOut

This operation transfers money from one account to another within the CGS. This is initiated by the user client effectively 'putting' money from their account into someone elses.

Input

- *DN*: String parameter containing the DN of the user account the money is to be transferred to.
- *Amount*: The amount of the currency that the client is asking to be transferred.

Faults

- *NoDestinationDNFault*: The DN the money is to be transferred to does not exist in the GBS.
- *NoSourceDNFault*: The DN the money is to be transferred from does not exist in the GBS.
- *NoMoneyFault*: Insufficient funds exist in the user's account to complete the transfer.
- *Fault*: Any other fault.

Only the holder of the account can initiate this operation. They are identified by the GSI authenticated connection. This operation must either be originated from the user (inconvenient) or allow a delegated proxy to perform the transaction (insecure?).

4.2.6 GBS::transferIn

This operation transfers money from one account to another within the CGS. The recipient is effectively 'getting' money from someone's account and placing it in their own.

Input

- *DN*: String parameter containing the DN of the user account the money is to be transferred from.
- *Amount*: The amount of the currency that the client is asking to be transferred.

Faults

- *NoDestinationDNFault*: The DN the money is to be transferred to does not exist in the GBS.
- *NoSourceDNFault*: The DN the money is to be transferred from does not exist in the GBS.
- *NoMoneyFault*: Insufficient funds exist in the user's account to complete the transfer.
- *Fault*: Any other fault.

This operation is essential for third party transactions but will expose the client (and the GBS) to potential abuse unless a secure mechanism for the user approving a transaction out of their account is achieved. The recipient of the money initiates this operation.

4.2.7 GBS::createAccount

This operation creates an account for the specified user with the stated amount.

Input

- *DN*: String parameter containing the DN of the user that the account is to be created for.
- *Amount*: The initial amount of the currency that will be in the account.

Faults

- *Fault*: Any other fault.

This operation is restricted to privileged users only.

4.2.8 GBS::deleteAccount

This operation removes the specified account.

Input

- *DN*: String parameter containing the DN of the user whose account is to be deleted.

Faults

- *NoAccountFoundFault*: The specified DN does not have an account in the GBS.
- *Fault*: Any other fault.

This operation is restricted to privileged users. In reality the account should probably be disabled as opposed to purged from the GBS.

4.2.9 GBS::createHold

During a long running execution it may be desirable to put a 'hold' on a certain proportion of a user's account to ensure that there is money left to pay for the consumed resources on completion. This operation creates a GBSHold instance that encapsulates the 'reservation' on the specified user's account. This port extends the Factory::createService port type.

Input

- *TerminationTime*: The earliest initial termination time that is acceptable to the client. This is effective the length of time that the client wishes to retain a 'hold' on the money. Must be specified.
- *HoldParameters*: This factory-specific element contains data used by the GBS Factory element to help instantiate a new service instance and set the cost for its use. The HoldParameters element MUST contain the following child elements:
    - *User:* This element specifies the DN of the account from which the specified amount of currency will be held.
    - *Amount:* Specifies the amount of money that is to be held.

Output & Faults
As defined in the OGSI specification document

4.3 Open Issues

4.3.1 Authorisation

There are many issues relating to authorisation that need to be resolved even within the protective 'sandbox' provided by the GSI container. For instance invoking a hold on someone's account should that require some authorisation of some sort? Should the user's proxy be presented along with the request to hold? Is this sufficient? Likewise with transfers some authorisation is needed to approve the transaction.

5 GBSHold Service

This service instance encapsulates the duration and amount of money being held on behalf of the client. OGSI standard lifetime management tools can be used to extend the length of this service (perhaps to a defined maximum from the GBS SDE).

[SJN: Would a notification subscription capability be of use here?]

The hold on the currency ends when this service instance expires or is terminated by the client. The service CAN only be terminated by the declared owner – the entity requesting the hold.

5.1 Service Data Elements

The current OGSI specification does not deal with any form of access control within SDEs therefore how much of the state encapsulated by the hold should be made public? If information were to be made public the SDE would be of the form:

```
<gesa:gbsHold>
  <gesa:account>
/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=steven newhouse</gesa:account>
  <gesa:amount>500</gesa:amount>
  <gesa:currency name="HeyPounds" email="cash@hey.ac.uk" />
  <gesa:owner>
/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=LeSC Broker</gesa:owner>
</gesa:gbsHold>
```

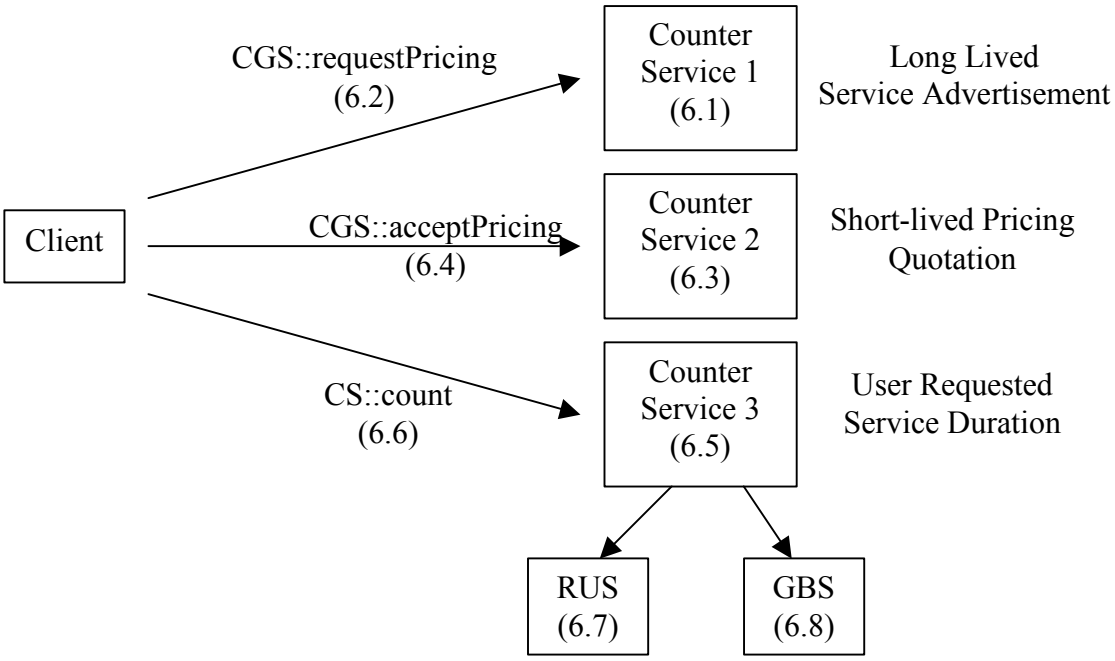
In this example the account is the Grid ID of the account holder, the owner of the hold is the entity requesting (and therefore controlling) the hold, and the amount (and in which currency) of the money from that account that is to be held.

5.2 Interface Definition

This service's primary function is to encapsulate the 'reservation' of money from a specified account. The amount encapsulated by the reservation is immutable therefore the only operations that may be performed on the service instance are effectively related to the lifetime management of the reservation (extended or terminated) and this may be restricted by the GBS during service creation.

6 An Example

To illustrate the previous service definitions we consider the simple economic use of a service to purchase the use of a service by a client, to record the use of the service within RUS, and to have the appropriate amount of money deducted from their account within a Grid Banking Service. The relationship between the client and the service instances is defined below.



The protocol communication passing between these entities is identified in the diagram and expanded in the following sections.

6.1 Economically Enabled Counter Service

The economically enabled Counter Service has the same basic interface as the Counter Service with the addition of a port to support the pricing of the service. This is described through the economic SDE's.

```
<serviceData name="gesa:economicSDE">
  <gesa:pricing name="FixedPrice" >
    <gesa:Duration default="30" maximum="60" />
    <gesa:chargedResources>
      <ur:invocation/>
    </gesa:chargedResources>
  </gesa:pricing>
  <gesa:currency name="HeyPounds" email="cash@hey.ac.uk" />
      …
</serviceData>
```

These SDE's (expiry information has been ignored) describe the economic state of the Counter service. The service is being offered for fixed price per invocation where the price in 'HeyPounds' will by default be held open for 30s before expiry.

6.2 Requesting a Price

From this information the client is able to compose a response describing the service invocation they require and this is passed to the `requestPricing` port:

```
<gesa:requestPricing>
  <gesa:pricingMechanism name="FixedPrice">
  <gesa:allowedUser>/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=steven
Newhouse</gesa:allowedUser>
  <gesa:allowedUser>/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=anthony
mayer </gesa:allowedUser>
  <gesa:currency name="HeyPounds" email="cash@hey.ac.uk" />
  <gesa:consumedResources>
    <ur:invocation>5</ur:invocation>
  <gesa:consumedResurces>
  <gesa:lifetime duration="180" />
</gesa:requestPricing>
```

This is a request to set up a service instance that is accessible by two users who estimate that they will use the service 5 times and are willing to pay in 'HeyPounds'. If this request is successful then a new instance of the service is created and a GSH is returned to the user. The service is expected to be used for 180s.

6.3 New Service Instance

The new service instance created by the previous activity results in a service instance with the following SDE's:

```
<serviceData name="gesa:economicSDE">
  <gesa:priceRate>
    <gesa:currency name="HeyPounds" email="cash@hey.ac.uk" />
    <ur:invocation>2</ur:invocation>
  </gesa:priceRate>
  <gesa:allowedUser>/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=steven
newhouse</gesa:allowedUser>
  <gesa:allowedUser>/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=anthony
mayer</gesa:allowedUser>
</serviceData>
```

The standard SDE structure will provide information relating to the service's expiry. The SDE's provides all the information relating to the service use specifying that 2 'HeyPounds' will be charged for each service invocation and is only accessible to the two specified users.

In creating the service instance there is an opportunity to perform several checks. A GBS can be found for the specified currency and the proposed users checked to ensure that they possess accounts and have sufficient funds to support the proposed charge for the estimated number of invocations.

6.4 Accept Pricing

The user is able to browse the SDE and view the offered service contract. This offer WILL be limited by the overall lifetime of the service as defined in the SDE and as requested by the user. If the user wishes to commit to the pricing they should invoke the `CGS::acceptPricing` port. This will create a new service instance for the user to interact with.

6.5 Grid Service Instance

This Grid Service instance is dedicated to the use of the stated user community at the previously quoted price. The user interacts with this service instance in the same manner as a non-GESA enabled service using the provided GSH.

6.6 Service Use

The uses the GSH to invoke operations in the Grid Service encapsulated by the CGS – in this case the `GS:count` operation. On each invocation the user access is check against the allowed users.

6.7 Resource Use

The consumed resources are recorded in an instance of the RUS. The consumed resources for this service would have the form:

```
<ur:UsageRecords>
  <ur:UsageRecord>
    <ur:GlobalUserID>/C=UK/O=eScience/OU=Imperial/L=LeSC/CN=anthony
mayer</ur:GlobaluserID>
    <ur:GlobalJobID>GSH of the invoking service</ur:GlobaJobID>
    <ur:Resource name="UseCount">1</ur:Resource>
  <ur:UsageRecord>
<ur:UsageRecords>
```

and be passed to the `RUS::insertUsageRecords` operation within the RUS service instance defined within the `gesa:resourceUsage` element of the CGS SDE.

6.8 Service Charging

Following service invocation (or on service destruction/periodically) the service use has to be charged for. All records relating to this service instance are retrieved from the RUS and the total cost for using the service calculated from the total resource usage and the stated charging policy. This cost is passed onto the CGS and any holds on the user's accounts released.

Usage information would be extracted from the RUS through the `RUS::extractUsageByGlobalJobID` operation specifying the GSH of the service. The Usage Record(s) returned by this search need to be aggregated and combined with the charging information to define a cost for the service use. This cost is used within the `GBS::transferIn` operation (invoked by the service provider) and specifying the DN of the entity that created this instance of the CGS as the entity that is to be charged for the use of the service.

7 Security Considerations

This document assumes the availability of the security provisions from the OGSI. There is a need to be able to specify access to services on a per user basis.

8 Outstanding Issues

There are many issues that still need to be defined:

- Continued definition of the SDE's – this will be ongoing.
- The current GESA SDE requires a declaration as to who is allowed to access a service. This has to be applied at a service level as opposed to current mechanism within OGSI which are focused at the service container. Mechanisms need to be developed to declare a service's availability on a user level and define access to the SDEs. This will require work with the OGSI-WG.

9 Acknowledgements

The creation of this document has been supported by:

- The Computational Markets project funded under the UK e-Science Core programme by the Department of Trade and Industry and the Engineering and Physical Sciences Research Council (http://www.lesc.ic.ac.uk/markets/)
- Other contributers…