DataGrid

AN ACCOUNTING SYSTEM FOR THE DATAGRID PROJECT

PRELIMINARY PROPOSAL- V 3.0

Document identifier:	**DataGrid-01-TED-0115-3_0**
Date:	**27/02/2002**
Work package:	**WP1**
Partner:	**INFN**
Document status	**DRAFT**
Deliverable identifier:	

Abstract:

The aim of this document is to propose a preliminary scheme and basis of discussion for the accounting to be used within the DataGrid Workload Management WorkPackage (WP1) starting from the `Computational Economy' model ([R3], [R4], [R6], [R10], [R11], [R12], [R13], [R14], [R15]), that we proposed for integration into DataGrid [R5].

The most interesting attribute of this economical approach is its capability of regulating resource usage of the Grid, that should ease the complex task of workload management.

We briefly discuss the problems of correctly choosing and quantifying the items to be charged. We describe the scheme of an implementation based upon the concept of Home Location Register borrowed from the GSM model.

We also try to address the problem of local accounts management on Grid resources, proposing the use of a system of dynamically created accounts called *template accounts* [R2].

Delivery Slip

	Name	Partner	Date	Signature
From	A.Guarise			
Verified by				
Approved by				

Document Log

Issue	Date	Comment	Author
2_0	27/09/2001	First public version	C.Anglano, S.Barale, L.Gaido, A.Guarise, S.Lusso, A.Werbrouck
2_1	31/10/2001	Revision due to F.Carminati dialog form	
3_0	01/03/2002	Modified to include broker interaction And a proposal for a monitoring service.	

Document Change Record

Issue	Item	Reason for Change

Files

Software Products	User files
Word	DataGrid-01-TED-0115-3_0-acct_prop.doc

CONTENT

1. INTRODUCTION 5
1.1. OBJECTIVES OF THIS DOCUMENT 5
1.2. APPLICATION AREA 5
1.3. TERMINOLOGY 6
2. THE “COMPUTATIONAL ECONOMY” MODEL CHOICE 8
3. RESOURCE ELEMENTS TO BE CHARGED AND ACCOUNTED 9
4. RESOURCE VALUE AND COST ALGORITHM 10
5. ACCOUNTING POLICIES 12
6. WORKING SCHEME 14
6.1. NOTATION 14
6.2. JOB SUBMISSION 14
6.3. AUTHORIZATION MECHANISM DRIVEN BY ACCOUNTING CONSIDERATIONS 15
7. LOCAL ACCOUNTS ON GRID RESOURCES 18
7.1. MANY-TO-ONE CORRESPONDENCE 18
7.2. ONE-TO-ONE CORRESPONDENCE 18
8. TEMPLATE ACCOUNTS 19
8.1. INTEGRATION WITH ACCOUNTING 19
9. INTERACTION WITH OTHER DATAGRID WPS 21
10. CONCLUSIONS 23
11. ANNEXES 24
11.1. HLR DATABASE SCHEMA 24
11.2. SENSORS 30

1. INTRODUCTION

1.1. OBJECTIVES OF THIS DOCUMENT

The starting material for this document is a wide selection of the pertinent literature and the objective is the definition of a suitable accounting architecture based on a computational economy model the choice of which is justified in section 2.

This preliminary work revealed the triple nature of the accounting problem.

The first issue, discussed in section 3, is how to treat the resource usage for which the user is charged.

The second issue is how to manage the Computing Elements so as to grant access (despite the limitations of the operating system) to a number of Grid users that, in principle, could grow indefinitely. To address this problem we are proposing the use of a system of *dynamical* Unix accounts known as *template accounts* and described in [R2]. This system is based on the idea to create some standard Unix accounts on a local machine and to link them dynamically to Grid users as the Resource Broker submits their jobs to the local resource.

The third issue, which is our principal subject of study, is how users can pay for computing services received; we decided to address this problem through the use of economic transactions between producers (the resources) and consumers (the users), within the context of an economic model. In this model, users pay in order to execute their job on the resources and the owner of the resources earn credits by executing the user jobs.

To address this issue we propose to use a Home Location Register (HLR) that is a sort of bank branch which manages the `accounts' containing the Grid credits of all users and all resources that refer to that particular HLR. This is similar to the mechanism used by the GSM mobile phone network to keep track of the cost of each telephone call made by the subscribers to a specific HLR.

1.2. APPLICATION AREA

Reference documents

[R1] B. Thigpen, T.J. Hacker - *Distributed accounting working group* - http://www.nas.nasa.gov/~thigpen/accounts-wg/index.html

[R2] T.J.Hacker, B.D.Athey – *Account - Allocation on the Grid* - http://www.nas.nasa.gov/~thigpen/accounts-wg/Documents/accounttemplates.pdf

[R3] L.F.McGinnis - *Resource Accounting, Current Practices* - http://www.nas.nasa.gov/~thigpen/accounts-wg/Documents/Current_Practices.pdf

[R4] F.Ygge, H.Akkermans - *Duality in Multi-Commodity Market Computations* - http://www.soc.hk-r.se/research/1997/dmcmc.ps

[R5] S.Barale - *The "Computational Economy" model applied to DataGrid Accounting* - http://www.infn.it/workload-grid/docs/20010409-barale-catania-account.pdf

[R6] R.Buyya, D.Abramson, J.Giddy - *A Case Economy grid Architecture for Service Oriented Grid Computing* - http://www.nas.nasa.gov/~thigpen/accounts-wg/Documents/ecogrid.pdf

[R7] A.Guarise, S.Barale -*Accounting: proposal for guidelines* – http://www.infn.it/workload-grid/docs/20010508-Accounting-guarise.pdf

[R8] S.M. Fitzgerald, G. von Laszewski, M. Swany - *GOSv3: A Data Definition Language for Grid*

Information Services. - GridForum Working Group Document GWD-GIS-011-5, Argonne National Lab. and Pacific Northwest Lab., Sep. 2001. http://www.gridforum.org

[R9] F. Pacini - *Job Submission User Interface Architecture* - http://www.infn.it/workload-grid/documents.htm

[R10] R. Wolski, J. Plank, J. Brevik, and T. Bryan. - *G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid.* - Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS2001), April 23-27 2001, San Francisco (CA).

[R11] R. Wolski, J. Plank, J. Brevik, and T. Bryan. - *Analyzing Market-based Resource Allocation Strategies for the Computational Grid.* - Technical Report CS-00-453, Department of Computer Science, University of Tennessee, Knoxville (TN).

[R12] R. Wolski, J. Plank, J. Brevik, and T. Bryan. - *G-Commerce -- Building Computational Marketplaces for the Computational Grid.* - Technical Report CS-00-439, Department of Computer Science, University of Tennessee, Knoxville (TN).

[R13] Rajkumar Buyya, Jonathan Giddy, David Abramson. - *A Case for Economy Grid Architecture for Service-Oriented Grid Computing* - 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), San Francisco, California, USA, April 2001.

[R14] Rajkumar Buyya and Sudharshan Vazhjudai - *Compute Power Market: Towards a Market-Oriented Grid* - The First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), Brisbane, Australia, May 15-18, 2001.

[R15] Rajkumar Buyya, Heinz Stockinger, Jonathan Giddy, and David Abramson. - *Economic Models for Management of Resources in Peer-to-Peer and Grid Computing, Technical Track on Commercial Applications for High-Performance Computing* - SPIE International Symposium on The Convergence of Information Technologies and Communications (ITCom 2001), August 20-24, 2001, Denver, Colorado, USA.

[R16] WP02: Grid Data Management - *Data Management Architecture Report* - http://grid-data-management.web.cern.ch/grid-data-management/docs/DataGrid-02-D2.2-0103-1_2.pdf

1.3. TERMINOLOGY

Definitions

Computational Energy	A quantity expressing the total amount of computation effort used by a job.
GridCredits	Currency used in the economic transactions between producers (the computing resources) and consumers (the Grid users) on the Grid.
HLR	DataBase that maintains the fund status of users and resources; it is also responsible for managing the economic transactions between producers and consumers. There will be many HLRs spread over the Grid, each HLR will manage a subset of the Grid users and resources.
Job cost	The cost of a job in GridCredit units (see section 4).
PA: Price Authority	An authority that can set the prices of the resources.
Resource value	An element estimating the “real” computing power of a resource element.
Resource price	The price assigned to a resource element starting from its value.
VO: Virtual Organization	An organization that administratively groups a set of user and/or resources.

Glossary

Economic model	A model that regulates the virtual economic transactions between all the entitities involved in the Grid.
Authentication	The phase in which a user and a resource mutually check each other's identities.
Authorization	The phase in which the system grants to the user access to the system
Accounting	The phase in which the system audits the usage of system resources.

2. THE "COMPUTATIONAL ECONOMY" MODEL CHOICE

The problem of the accounting on a computational resource can be faced in many different ways.

The aim of this section is not to describe the details of all these possible approaches (on which an exhaustive literature exists) but to propose our vision of the problem.

We think that a promising way of solving the problem of resource allocation in a Grid environment is an accounting procedure based upon a computational economy model.

This choice comes from the fact that this model, once a valid price setting policy has been established, should lead to a state of nearly stable equilibrium able to satisfy the needs of both resource `producers' and `consumers'. By `equilibrium' we intend that exchanges between entities that both supply and use computer resources is such that usage supplied at one time is recovered later without significant loss or gain.

Moreover this model should provide a *self-regulation* of the workload.

An economic model requires, for its application, a monetary or credit unit. We propose a type of Grid Credit easily related to the predominant factor in computer usage cost, for example, integer or floating point SPEC units. This concept is clarified in section 4.

An accurate analysis of the many reasons in favour of this approach instead of a simple `passive' logging-like model can be found in [R1], [R2], [R5] and [R6].

In addition we recall the fact that Accounting and Authorization are tightly bound in our model. In other words the Authorization to use a resource is conditioned by the availability of credits in the HLR of the user, thus encouraging him/her to sometimes choose a less expensive available resource.

Experiments in this direction are being performed by some already active Grid testbeds like: NASA-ames, PSC, PNNL and Wright-Patterson[R3].

3. RESOURCE ELEMENTS TO BE CHARGED AND ACCOUNTED

It is necessary to decide for which resource elements one should pay.

In [R3] an analysis of the elements accounted in the four already mentioned testbeds is reported. They are:

- **NASA-ames:** CPU+memory+wallclock
- **PSC:** CPU+memory+connect
- **PNNL:** CPU
- **Wright-Patterson:** Wallclock*(# CPUs)

It is clear that every site has decided independently from the others what, how much and when to "charge" for the resource usage. This could also be the case of DataGrid, but we strongly support the attainment of a common agreement on the resource elements to be charged.

We think that, in order to take care of most needs, one should consider at least the following elements:

- priority in a batch queue,
- cost per cpu-time unit (or equivalent in some sort of benchmark),
- cost per wall-clock time,
- cost for memory usage,
- cost for disk storage occupation,
- disk swap availability,
- network data transfer cost,
- Tape storage cost (if applicable).

Obviously not every resource will bill the cost of every chargeable element, but only those which it offers and considers chargeable.

We would like to note that in [R1] the authors describe the eventuality of billing even some aspects that are very difficult to quantify, like, for example, the cost for a local consultant or a programmer needed to control the execution of some difficult jobs, or the transport cost of the data via some removable media, like tapes or other storage devices. For simplicity we think that such aspects should be neglected at this stage.

Concerning the elements that we should charge for, there are many open issues, both technical and "political": we have to define well which of the mentioned items should be charged to the users, and which ones could be neglected. Then we have to correctly define how to implement the counters for each charged element (See 11.2).

4. RESOURCE VALUE AND COST ALGORITHM

One of the most important open issues is to determine the *value* for every resource element and consequently, the price. The price together with the amount of usage of the resource determines the cost.

It's important to remark that the value and the price of a resource are conceptually different.

The value of a resource should essentially express the real contribution of the resource to the computation. We assume that the price of a resource should be related to the value of that resource and that two resources with the same value should, a priori, have comparable prices, while this is not necessarily true in real life economy.

We can imagine different 'economies' in which the relation between price and value follows different laws, leading to variable behaviours.

Even if our implementation should permit the possibility that value and price be totally uncorrelated, we think that, inside DataGrid, there should be a common algorithm that correlates the value of a resource and its price. We feel that such an algorithm will help to maintain equilibrium as we have defined it.

It's clear that, in order to define this algorithm, it is therefore necessary to have a general way to estimate the value of a resource.

One way to do this is to properly relate the value of a resource to the performance it delivers to applications. This value can then be estimated by defining a suitable set of benchmarks covering the pertinent aspects of the computation. Whenever a resource is connected to DataGrid, the benchmark suite is executed on it.

It is important that the benchmark results must not be falsified by a malicious resource's owner to alter the value of his/her resource, as every user has to be certain to have really paid for what was actually obtained (i.e., prices must be fair), in order to maintain equilibrium.

This can be obtained by installing the benchmark suite on each resource like a sealed *black-box*, checksummed with some sort of hashing algorithm, like for example the MD5. The fingerprint of the black-box should be publicly available, so everyone can check at any time that the black-box on a resource has not been altered, this can be done (e.g.) by sending directly to the "suspect" resource a job that checksums the benchmark suite.

The cost algorithm is an automatic procedure that calculates the price of the resource, as a function of the characteristics of the resource elements, the user's resource usage and the economic model adopted.

We define computing usage as the product $p \cdot u$ where p is a performance factor or power (i.e. the benchmark) and u is the amount of usage of that resource element. For example if p refers to the CPU power, u should be the amount of CPU time used by the job.

Ideally, in this case, this product that we will call *technical cost* should be nearly constant for a given job executed on processors with varying CPU power. Basically this technical cost could even be referred to as a *computational energy* in the sense of being the product of power and time.

For what concerns the cost algorithm for the whole job, it could be obtained from the technical cost $p_i \cdot u_i$ of every resource component by computing:

$$P = \sum_{i=1}^{N} k_i \cdot (p_i \cdot u_i) \cdot w_i$$

Where p_i and u_i are defined above and w_i is a generic element of a vector of weight factors used to arbitrarily adjust the price of the resource components according to the chosen economic model. The *i* index runs over the resource elements (i.e. CPU, RAM, Disk, etc…).

The normalization coefficient, k_i, is used to convert the measurement units of the formula into GridCredits. Since *P* must be expressed in Grid Credits (*G*), it is necessary that every term under the sum sign be expressed in Grid Credits too.

So if we analyse the measurement units in the formula we have, for the known factors:

$$[p_i] = P \quad \text{Performance Factor}$$
$$[u_i] = U \quad \text{Usage Factor}$$
$$[w_i] = A \cdot A^{-1} \quad \text{Weight, adimensional}$$

So, to obtain Grid Credits for *P*, we have to introduce k_i as measured in:

$$[k_i] = G \cdot P^{-1} \cdot U^{-1}$$

In this way we have:

$$[P] = [G \cdot P^{-1} \cdot U^{-1}][P][U][A \cdot A^{-1}] = [G]$$

For example, if we are talking about CPU usage we may have:

$$[p_i] = SpecINT$$
$$[u_i] = t$$
$$[k_i] = G \cdot SpecINT^{-1} \cdot t^{-1}$$
$$[w_i] = A \cdot A^{-1}$$

Thus resulting *P* expressed in Grid Credits.

The economic model is still an open research issue that we plan to investigate further. For this reason, we have kept our working model general enough so that it will be usable regardless of the particular economic model that will be adopted. As a matter of fact, an economic model determines the way the w_i values are set. For instance, in the “supply and demand” model, a resource owner might autonomously set the values of the w_i vector for his/her own resources in order to attract or deter external users. This model is clearly very simple to implement, but almost impossible to manage in order to reach some form of equilibrium. Conversely, an economic model based on “general equilibrium” requires that prices be set by an independent entity (a Price Authority) that acts for the good of the whole market, and not of individual resource owners. The Price Authority would have to use a predefined decision scheme upon which Grid users agreed. For example, if under that PA there are four resources, the system can monitor the number of job waiting in queues for each resource. If one of these resources (which we suppose identical in terms of value) has fewer queued jobs than the others, the PA should diminish the weight factors of that resource in order to attract the submission of new jobs on it. Then, as the queues become balanced the weights are readjusted.

Clearly this is a very strong simplification of the problem, but should be enough to show how the economic system and this implementation can be used to help in balancing the workload on the resources.

5. ACCOUNTING POLICIES

Once the accounting will be implemented, it will be necessary to define the rules of the system.

There will be basically two type of rules, first the administrative ones, that covers some general aspects of the system and will deeply influence the behaviour of the whole grid and, secondary the system policies that should cover only some "particular" aspects.

In our model (see section 6), the HLR of the user pays the consumed credits to the HLR of the resource. The accumulated credits at each resource HLR should be periodically redistributed to the users belonging to that HLR.

It may happen (and in the first times it will) that a user in order to complete a job will need to make some debts, so in the HLR database we have to insert some information about the debt limit of the user.

For these reasons in the beginning there should not be a debt limit, to let the user become familiar with the system, and in practice, the accounting will only report back the resource consumptions and cost but not debit the accounts.

Another policy concern is how to adjust accumulated differences between credits and debits. These can arise in three cases:

- **Producers only:** Such as a computer centre. Grid credits need to be converted in current currency units and invoiced to the users as such.
- **Consumers only:** Such as a small biotech laboratory. Need to convert real currency into grid credits.
- **Producers/Consumers:** Such as a common physics group. These can negotiate when and how to absorb credit-debit differences.

Intrinsic in the first two cases is the need for the definition of a conversion mechanism between grid credits and common currency.

It's clear that these are decisions that should be defined at the management level.

According to our point of view we propose the following system policies in the following particular situations.

- **Job cancelling during execution (due to user action):** The user is billed for the work already done, so every user is incentivated to use the Grid only when really certain about what the job actually does.
- **Job cancelling in a resource queue (due to user action):** No charge to the user, this event already penalizes the user due to loss of position in the resource job queue.
- **Remote resource crash:** No charge to the user, so the local administrator is incentivated to maintain high resource reliability.
- **User job crash:** In the production phase the user is charged for the resources used.
- **Job energy underestimation:** The job is executed till the user has enough credits left, but he/she will be penalized on the next job run (see section 6.3).
- **Job energy overestimation:** The system will slightly penalize the user on the next job submission (see section 6.3).
- **Job energy not esteemed:** The user can submit only one job per time.

One of the most important things we want to point out is that, in the earlier stages of their work, users will need to familiarize themselves with the accounting without the fear of spending all their credits by mistake.

To address this problem some parameters could be set temporarily in a different way from the real production phase.

An accounting system based on a computational economy model should be simple enough to be understood by all producers and consumers. In fact the final user should only have to specify a few parameters using the *JDL* concerning the amount of credits expendable for the job and an estimation for the computational energy required, the rest of the work should be done transparently by the system.

6. WORKING SCHEME

In this section we present the working scheme of our accounting model, in order to explain the mechanism of economic transactions using the *HLR* system.

We want to stress that the implementative model of our system is still independent from the economic model and from its associated policies. Both the economic model and the system policies are still mainly to be chosen, and the DataGrid project will provide the experimental ground to make and improve these choices.

6.1. NOTATION

In order to correctly describe our accounting system we will use the following terms:

- **K(i,j):** j-th entity associated with the organization i, K stands for one of the following types of entity.
    - **U:** User.
    - **R:** Computing Resource.
    - **J:** Job.
- **HLR:** Home Location Register.
- **RB:** Resource Broker.
- **IS:** Grid Information Service.
- **JSS:** Job Submission Service.

Example: U(5,3) stands for the user number 3 of the organization identified by the id 5.

Now we can examine the working scheme of our system when a user submits a job to the grid.

6.2. JOB SUBMISSION

- **A:** When the user U(i,j) submits a job to the RB using the JDL specification language, the request should contain those parameters needed by the RB in order to correctly choose the set of resources, and from those resources to estimate the job cost. Some of these parameters may be:
    - Estimated computing energy needed.
    - RAM and mass storage needed.
    - Maximum expense.

  and others as needed.
- **B:** The Information Service provides to the RB a complete list of resources that satisfy the job request. Then the RB gets the price list for those resources by the Price Authority and uses a Cost Estimation Algorithm to foresee the job cost. The estimation will be done according to the information given by the user with the JDL.
- **C:** The RB asks to the HLR of the user information about the fund availability.
- **D:** The HLR answers back to the HLR whether the user has enough fund or not. If the answer is affirmative, that amount of fund are locked.

- **E:** Now the RB has all the information needed to chose the "best" resource for that job. The choice is made both upon technical and economic parameters. The job can then be submitted via the JSS to the Computing Element.
- **F:** The job runs on the Computing Element and is continuously monitored by the Monitor Service (see 11.2).
- **G:** Information gathered by the Monitor Service are periodically sent to the User HLR. On the User HLR the Cost Algorithm uses these information to compute the partial cost for that job. If the cost exceeds the amount of fund assigned to that job, the HLR halts the execution of the job.
- **H:** Once the job finishes (satisfactory or not) the user HLR computes its final cost. Finally the user HLR sends to the Computing Element HLR the credits spent by the job and unlocks the residual of the funds.

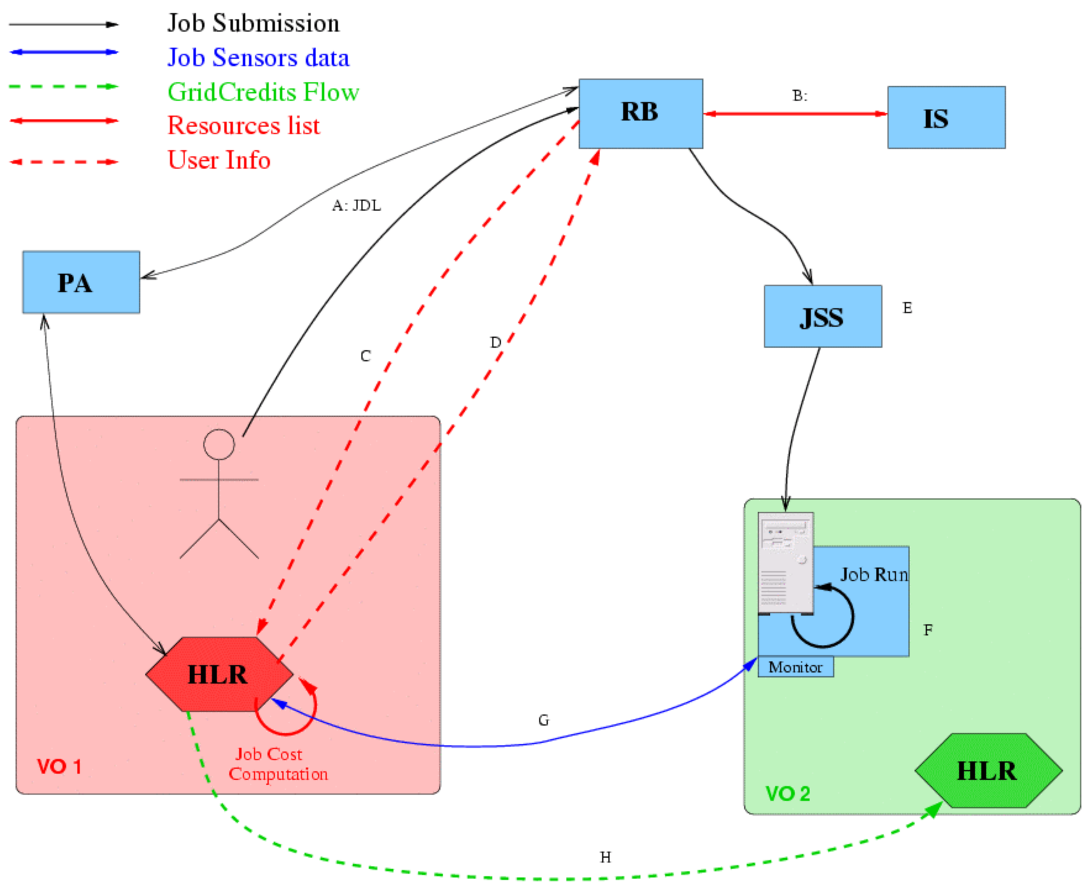

Figure 1: Job submission scheme

6.3. AUTHORIZATION MECHANISM DRIVEN BY ACCOUNTING CONSIDERATIONS

In our vision the Grid will be ruled by an economically modelled accounting system. It is therefore necessary that the authorization to use a resource come directly from the user's fund availability. In the following we examine two feasible approaches.

In the first the user submits as part of the JDL an estimation of the computational energy needed by the job. The resource, knowing its value and its current price, uses the cost algorithm to compute an estimation of the job cost. The cost is then communicated to the user HLR requesting the reservation of the needed funds. At this step the user HLR replies to the resource whether or not the user has enough funds and can run the job. Once the job has finished the "true" job cost is computed and the invoice is sent to the user HLR.

In this approach, to avoid user speculation, we can propose to penalize the users that give wildly wrong job estimations by increasing the price of the resource (for that job only) if the deviation from the given estimation exceeds a reasonable amount.

The second approach could be to check the consistency of the user's funds before authorizing access to the resource. If there are still some available credits then the user is authorized to run the job (without having to forecast its energy necessity). However it is necessary to reserve a fixed amount of credits before launching the job to avoid the possibility that a user runs thousands of jobs before being blocked by the system.

This approach has the advantage that it is not necessary to give a job cost estimation. However it has the disadvantage that a user can spend with an authorized job more than the available credits. It is therefore necessary to manage the user debts. It is clear that with this approach a user can successfully complete a job even if doing so creates some debts, but then the user can not run any other job until his/her account has returned positive.

Even if this is not clear at first, these two approaches are not incompatible. In fact we can imagine a grid environment in which if the user gives an estimation of the computing energy needed by the job the first approach is used. Otherwise, if the user is in a group that can absorb debts, the second method can be used.

We assume:

1. that pricing policy will take into consideration the depths of queues with other parameters in order to facilitate load balancing.
2. that the system will accept jobs with or without energy estimates.

We then arrive at a proposal for the management of the reservation of GRID credits which has the following objectives:

- To encourage accurate estimate of computational energy for a job.
- To allow first-time users to enter cautiously into the GRID environment without disturbing the experienced users.

The proposal is the following:

A user who claims to have insufficient knowledge of the requirements of his/her job submits it with no estimate of needed resource usage. This job is accepted if the user's account shows a positive balance and no other job without estimate is currently submitted. When this job is completed the amount of computational energy is communicated to the user who then has a basis for estimating the usage requirements of future jobs of this type.

A user who furnishes estimates for the job requirements can submit a job before the previous one completes. Each job with an estimate entails the reservation of the user's funds corresponding to the estimate. An account can also show a negative balance if there is a non-zero debt limit. If the account has a negative balance and there is not a finite debt limit capable of covering the estimate, the job is refused. There are two relations between estimates and usage costs. The estimate is either greater than or less than the actual cost. The consequences of these two cases are different. If the estimate is greater than the cost, the user is somewhat penalized in having reserved more credits than necessary and the depths of the queues were falsified leading to a perturbation to the load balancing procedure.

If the estimate is less than the cost, there is the danger that the user will create debts greater than the debt limit which is unfair to the other users.

To avoid excessive stress in job cost estimation, the reservation management foresees a percentage tolerance (TOL) within which cost and estimation are considered equal.

In both cases the applied costs are calculated in the same way so that the reservation management will not introduce inflation. Thus it is necessary that highly wrong estimates have an effect on the next job submission.

In the case of under estimates, a percentage:

$$PC = \frac{Cost - Estimate}{Estimate}$$

is applied to the next job requiring a reservation of an amount:

$$Amount = (Estimate) \cdot (1 + PC - TOL)$$

If the estimate and cost are equal for this job within the tolerance, PC is zeroed and no difference between estimate and reservation is applied to the next reservation.

In the case of over estimates:

$$PC = \frac{Estimate - Cost}{Estimate}$$

Here eventual penalties should be much lighter, sufficient to avoid wild estimating.

A proposal could be

$$Amount = (Estimate) \cdot \left(1 + TOL - \frac{PC}{10}\right)$$

which for wild overestimating could result in an under estimate and the previous path to sufficiently accurate estimating.

Using the accounting to drive the authorization procedure leads to split the authorization mechanism in two layers.

In fact we can see the authorization structure as made of two layers:

1. User authorization per resource.
2. User authorization per job.

In the first layer the system managers, according to whatever policy they decide, authorize a set of users to run their jobs on a particular resource, or set of resources.

In the second layer, the accounting system, decides if a particular job of an already authorized user can be executed on a resource. This decision is taken, by the accounting system, upon the fund availability of the user and his past behaviour.

7. LOCAL ACCOUNTS ON GRID RESOURCES

From the point of view of system planning and effective accounting, one of the most important administrative issues on a Grid is the management of local accounts on each resource.

In the literature (see [R1], [R2] and [R4]) we found two antithetical approaches to the management of local accounts:

- many-to-one correspondence
- one-to-one correspondence

We think that both these approaches, while presenting the unquestionable advantage of being easily realized, have the remarkable disadvantage of limiting strongly the scalability and flexibility of the system. In the following sections we will analyse them briefly to point out these contraindications.

The Globus software, via the *grid_mapfile*, has the capability to manage both approaches.

7.1. MANY-TO-ONE CORRESPONDENCE

Th44e first approach is to map many Grid users to a single static Unix account, using their credentials (e.g. an X.509 certificate).

This approach has the strong contraindication of requesting an intensive administrative work, since managing the *grid_mapfile* requires much work for each machine.

Another problem, from the accounting point of view, is that tracking the resource usage performed by users could become very complicated when mapping every user to the same static Unix account.

7.2. ONE-TO-ONE CORRESPONDENCE

This approach seems to allow an accurate tracking of users and jobs usage of the resources. But in principle, and probably even in practice, the same user could have different login names on different resources, making it almost impossible to identify him univocally.

In addition the problem of scalability when administering a large number of accounts seems to strongly question the validity of this approach.

8. TEMPLATE ACCOUNTS

It seems clear that none of the above approaches will be able to satisfy our needs in the long run.

Two alternative (and similar) approaches to the problem of account management on Grid resources, that seems to meet the requirements of accounting and system management, are described in [R2] and [R4] with the names of *template accounts* and *virtual accounts*.

Both models are based upon a link of a credential of the Grid user(e.g. the X.509 certificate) with a *dynamic* Unix account, where dynamic means that this account is (re)assigned from time to time according to the requirements of the users and the system.

The solution described in [R2], in our opinion, seems to be able to obtain a coherent, functional and scalable Authorization Authentication and Accounting (AAA) structure. This approach basically represents a compromise between the two polices: one-to-one and many-to-one. In addition it manages the persistence of frequently used accounts which strongly reduces access time for frequent users, while keeping it at an acceptable value for new users.

Another aspect that has been analysed in [R2] is the possibility to correctly decide the number of available accounts on a resource on the base of a preliminary analysis of its workload, but we won't go into further details here because they're fully treated in the reference.

8.1. INTEGRATION WITH ACCOUNTING

We now describe how *template accounts* relate to our proposed accounting model.

The basic idea is to create a set of *template accounts* on each resource.

These accounts should not be permanently linked to a physical Grid user: a temporary link is created each time an authorized user needs to use the resource.

This means that every *template account* should use the normal attributes of a Unix account, but the logging information should refer to a virtual user instead of a real one.

In [R2] the possibility of an interactive login on the remote resource for an authenticated user is discussed. To perform this task the authors suggest to use something like an SSH adapted for accepting X.509 credentials. This feature can, in our opinion, be useful in the case of interactive jobs, but it is not necessary for batch ones.

Now we'll outline the different phases of the login process (see Figure 2) on a resource using *template accounts*, considering only the batch sessions (for interactive ones refer to [R2]).

- The Resource Broker issues a submission request to the resource sending a valid credential (like an X.509 proxy credential). From the accounting point of view the submission request should contain the location of the user HLR. The last is still an open issue; we think that a good way could be inserting the URL of the HLR into the authorization certificate subject; another feasible approach could be the a priori binding of the user's HLR to his/her email address (e.g. guarise@to.infn.it should have its HLR in hlr.to.infn.it).
- A task on the resource checks the validity of the user credentials (via the GSI protocol). If these are valid the user is authenticated, otherwise the job is rejected and the RB informed about the rejection. In both cases the events should be recorded by the Logging and Bookkeeping service.
- Once the user has been authenticated the authorization task starts. It should be done by the economic modelled accounting system that has been presented in 6.3.

- Once the authentication and authorization steps are completed, the system links the Grid user to a *template account*.

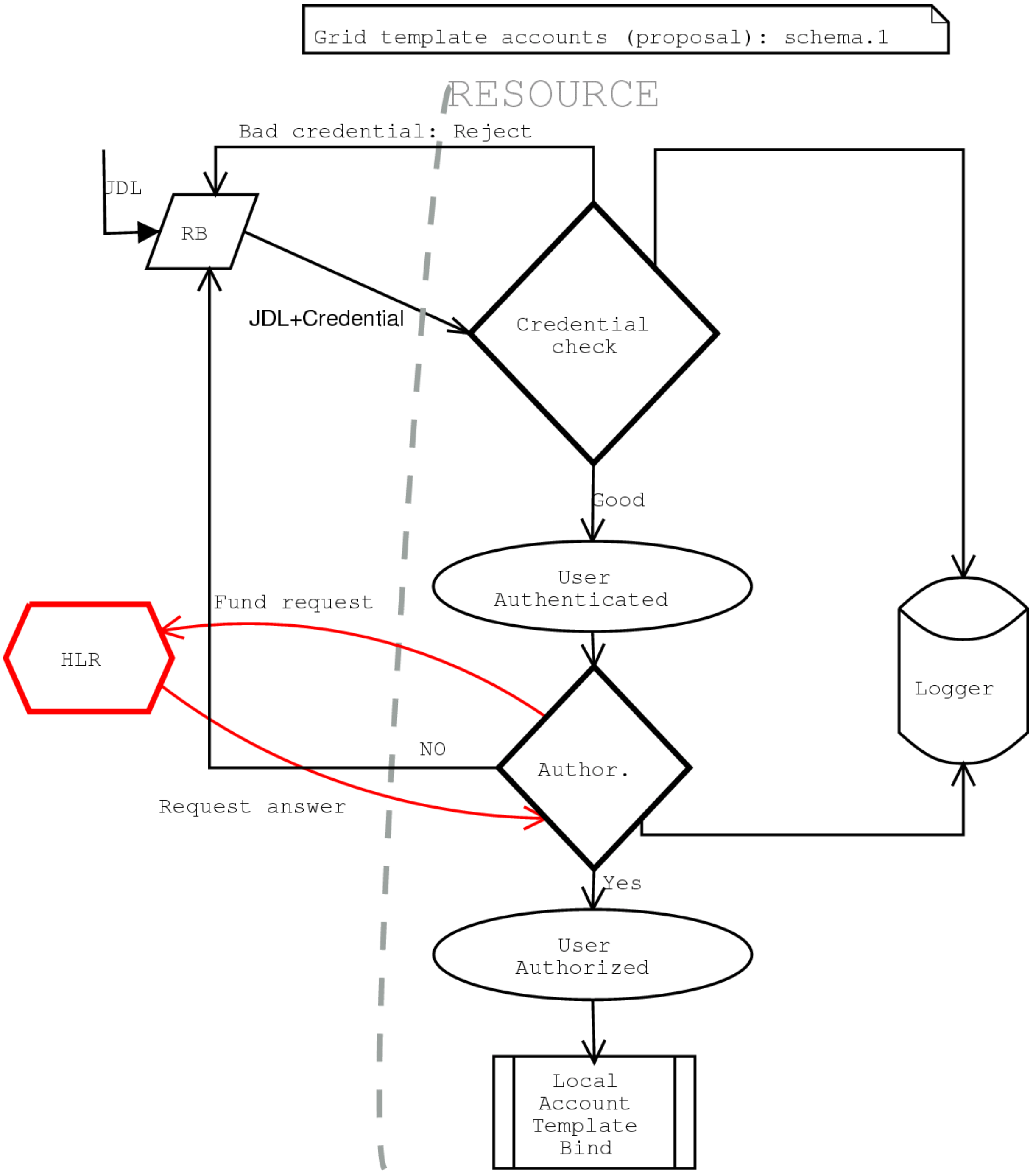

Figure 2: Authentication and Authorization

9. INTERACTION WITH OTHER DATAGRID WPS

The problem of accounting is quite complex and has several aspects that are spread over many workpackages in the EU-DataGrid project.

While other WPs have expressed their requirements on accounting in the documents they distributed, also the accounting system has some requirements towards the other WPs. They will be described below.

The definition of the correct interfaces between the accounting system and the other DataGrid components is a continuing activity.

One of the main issues of the DataGrid project is the security infrastructure (Authentication, Authorization and Accounting).

The basic assumptions regarding security are:

- the use of X509 certificates issued by a Certification Authority together with GSI (Globus Security Infrastructure) for authentication,
- the use of delegation mechanisms through proxy certificates
- the need for secure data transfer
- the need for different levels of authorization (ranging from Virtual Organizations to single users)

While for the first three items we think we can apply already established grid practices without particular difficulties, the complex problem of authorization has not been deeply investigated yet.

An Authorization Working Group has been recently formed inside DataGrid with the aim of proposing an authorization mechanism for the whole project. At the moment they are evaluating the major issues and the possible solutions (including authorization certificates and the way to make them available to the community, e.g. CAS). We think that a close collaboration with the Authorization Working group has to be established.

Another important issue is the selection of the database to store the information relevant to the accounting system.

We think that the information about the economical transactions of the accounting model should be "classified" and managed in a different way from other types of data in order also to satisfy a specific requirement by WP8-10.

For this reason we think it would be preferable to use a dedicated database architecture infrastructure for such data. The possibility of using the general purpose SQLDataBaseService [R16], developed and proposed by WP2, has to be investigated.

Furthermore, there are some other functionalities requested by the experiments (WP8-10) concerning:

- Cost estimation: the end user needs some means to evaluate in advance the cost of the computation (provided via some appropriate cost estimators).
- Anonymous access: the possibility to have a restricted access to the resources with a different accounting procedure.
- Privacy of the HLR information: access to these data should be granted only to their owner and his/her V.O. Local site managers should also have access to data, but only for the purpose of incident tracking.

While the last requirement is included in our schema, we think that the first one is very difficult (if not impossible) to implement in an efficient and useful way; the second one has not been considered yet.

Some accounting related requirements to other WPs exist. We'll describe them briefly in the next paragraphs.

The first requirement regards the information about the resource consumption by the jobs. We need this data to properly calculate the amount of grid credits used by each job and to manage them in the HLR infrastructure.

Although it has not yet been decided which elements (CPU, memory, disk space, etc.) of the Computing and Storage Elements will be charged, we assume that someone has to provide us with such information.

Another issue is the management of user accounts on the Computing Element.

The accounting system deeply relies on the way in which the mapping between user accounts and grid accounts is accomplished, as we have described in chapter 7. To address this problem we investigated the use of the “template accounts” (chapter 8) and they seem to be a promising solution.

However other different solutions may be considered, and integrated with the DataGrid fabric management (WP4) environment.

10. CONCLUSIONS

We would like to end up with some considerations about the work done up to now and what has still to be done.

As highlighted in this proposal, the accounting is a complex system that involves many components.

To test the validity of our approach we started developing a prototype in order to demonstrate the feasibility of the basic functions of the HLR system, as well as to show many of the foreseen characteristics presented in this proposal.

The core of this prototype can be considered as an alpha version of the HLR management software.

Although the prototype covers just some specific aspects of the accounting system we think, that the economical approach with an HLR architecture (borrowed from mobile phones network systems) is promising.

Anyway many issues are still open and need further investigation.

In particular:

- which elements should be considered for charging (once these will be identified we showed how to compute the total cost of a job taking into account both performance factors and market policies);
- we have to investigate the problem of the cost estimation and its integration with the fund adequacy check;
- we have to address the problem of having a proper mechanism for treating guest users on a computing element (template accounts?);

Moreover the interaction of this system with the other DataGrid components needs to be analysed in further details, in a joint effort with the other work packages.

11. ANNEXES

11.1. HLR DATABASE SCHEMA

Many questions about the specifications are still open: we are considering to use a relational database, implemented according to the ANSI SQL 92 standard. For the first prototype it is MySQL v. 3.23.38.

In the following we'll describe the objects that should compose our database using the GOSv3 (Grid Object Specification version 3) standard[R8].

- **udesc:** this object contains the UserID (uid) of the users that refer to a specific Home Location Register (HLR) and their full name (e.g. uid: guarise, desc: Andrea Guarise). We suppose this object won't be more than some thousands of records long.

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS udesc {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing all users referring to a specific HLR
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY uid          ::single,ces;
         MUST CONTAIN {
             email        ::single,cis;
             descr        ::multiple,ces;
             cert_subject ::multiple,ces;
         }
    }
}
```

- **grdesc:** this table contains the GroupID (gid) of the groups to which users refer (e.g. alice, cms ...) and their full description (e.g. gid: alice desc: ALICE LHC experiment). We suppose this object won't be much long.

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS grdesc {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing groups to which users belong in a HLR
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY gid      ::single,ces;
         MUST CONTAIN {
             descr    ::multiple,ces;
         }
    }
}
```

- **fdesc:** this object contains the FundID (fid) of the funds that will provide grid credits for the users (e.g. alice, cms ...) and their full description (e.g. gid: alice, desc: ALICE experiment). We suppose this object won't be very long.

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS fdesc {
         OID {T.B.D.}
         DESCRIPTION{
             An object containing all funds that provide `grid credits' to
users
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY fid       ::single,ces;
         MUST CONTAIN {
             descr     ::multiple,ces;
             total     ::single,numeric;
             spent     ::single,numeric;
         }
    }
}
```

- **gf:** (grid funds) this object contains the balance of grid credits in terms of funds (fid) for each group (gid).

The content of field `total` represents the total grid credits assigned at the beginning of each accounting period to a group (gid) from the fund (fid) and can't be modified by means of automatic procedures.

The field `booked` contains the credits that are being reserved for each fid+gid pair, and is incremented every time the HLR accepts to pay for the execution of a job.

The field `spent` contains the sum of the credits already spent. Each time a job finishes the HLR frees the amount locked for the job and adds the sum to this field.

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS gf {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing the financial balance of funds
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY fidgid     ::single,fidgid;
         MUST CONTAIN {
             total      ::single,numeric;
             booked     ::single,numeric;
             spent      ::single,numeric;
         }
    }
    OBJECT CLASS fidgid {
         OID {T.B.D.}
         DESCRIPTION {
             A complex KEY object for the gf OBJECT CLASS
         }
         KIND {AUXILIARY}
         INHERITSFROM {DataGridTop}
         MUST CONTAIN {
             fid    ::single,ces;
             gid    ::single,ces;
         }
    }
}
```

- **ug:** (users/groups) this object contains the balance of funds (fid) for each group (gid) assigned to each member of the group (uid). The fields `reserved` and `spent` keep the meaning given above while `assigned` contains the total credits assigned to each user. When the job is completed the amount reserved is sent to the HLR of the resource that elaborated the job, the object *gf* is updated and a notification (e.g. via e-mail) is sent to the user.

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS ug {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing the balance of funds
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY giduid    ::single,giduid;
         MUST CONTAIN {
             assigned  ::single,numeric;
             booked    ::single,numeric;
             spent     ::single,numeric;
         }
    }
    OBJECT CLASS giduid {
         OID {T.B.D.}
         DESCRIPTION {
             A complex KEY object for the ug OBJECT CLASS
         }
         KIND {AUXILIARY}
         INHERITSFROM {DataGridTop}
         MUST CONTAIN {
             gid       ::single,ces;
             uid       ::single,ces;
         }
    }
}
```

- **transactions_out:** this object contains the list of all payments made by the user HLR to the resources one. Fields in this object are:
    - `tid`: identifies unambiguously the transaction
    - `uid`: identifies the user responsible for that transaction
    - `fid`: identifies the fund to debit for transaction
    - `amount`: is the amount of the invoice
    - `supplier`: is the identifier of the resource HLR
    - `tr_date`: is the time stamp of the transaction

This object is updated each a time job is completed and the credits are sent to the HLR (supplier) of the resource.

The problem related to this object is that its length is going to increase excessively.

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS transactions_out {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing the transactions exiting from the HLR
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY tid       ::single,numeric;
         MUST CONTAIN {
             uid       ::single,ces;
             fid       ::single,ces;
             supplier  ::single,ces;
             amount    ::single,numeric;
             tr_date   ::single,ces;
             dg_jobid  ::single,ces;
         }
    }
}
```

- **transactions_in:** this object contains the list of all payments made by remote HLRs to the local HLR. Fields in this object are:
    - tid: identifies unambiguously the transaction
    - fid: identifies the fund to increase after transaction
    - rid: identifies the resource that performed the calculation
    - amount: is the amount of the invoice
    - supplier: is the identifier of the HLR that paid the transaction
    - tr_date: is the time stamp of the transaction

This object is updated each time a job is completed and the credits arrive to the local HLR. The problem of this object is that its length is going to increase excessively.

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS transactions_in {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing the transactions entering the HLR
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY tid       ::single,numeric;
         MUST CONTAIN {
             rid       ::single,ces;
             fid       ::single,ces;
             supplier  ::single,ces;
             amount    ::single,numeric;
             tr_date   ::single,ces;
             dg_jobid  ::single,ces;
         }
    }
}
```

- **rdesc:** this object contains the list of all resources referring to the local HLR. Fields in this object are:
    - `rid`: identifies unambiguously the resource
    - `desc`: a brief description of the resource

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS rdesc {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing the description of resources belonging to
a HLR
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY rid          ::single,ces;
         MUST CONTAIN {
             descr        ::single,cis;
             cert_subject ::single,ces;
```

```
          }
     }
}
```

- **rgf:** this object contains the links between resources and funds for each group of grid users. Fields in this object are:
    - `rid`: identifies unambiguously the resource
    - `gid`: identifies the group that owns the resource
    - `fid`: identifies the fund where the credits earned by that resource should be put
    - `total`: is the total amount of credits earned by the resource

```
NAMESPACE DataGrid {
    DESCRIPTION {
        This GOS schema describes objects used by the DataGrid project
        }
    OBJECT CLASS rgf {
         OID {T.B.D.}
         DESCRIPTION {
             An object containing the links between resources and funds
         }
         KIND {STRUCTURAL}
         INHERITSFROM {DataGridTop}
         KEY rid       ::single,ces;
         MUST CONTAIN {
           gid       ::single,ces;
           fid       ::single,ces;
           total     ::single,numeric;
         }
    }
}
```

11.2. SENSORS

It should be clear that the accounting system needs to know many information about the user’s job in order to correctly estimate their cost. The cost of a job is computed from its resources usage, so it is important that have on every Computing Element some sensors that can report to the accounting system the resources usage for each job.

Many type of sensors are needed, one for each computing resources that we want to account like for example: CPU time, RAM allocation, Disk Usage etc… (See Par.3).

These sensors must, however, share a common set of characteristics and functionalities that now we want to explain.

First, we need an API to gather information about the real physical owner of a job, this can be summarized by the need of two functions used to get the CE,UID,PID (or list if not univocal) from the dg_jobid of a job and vice versa.

Every sensor must be a lightweight application in order to not interfere too much with the computation, they must take in input the Unix PID of the running process to monitor, and return both the punctual and integral values of the information monitored.

We can now explain how we intend to use these sensors in the accounting environment.

Suppose we have the following tools:

- **dg_jobid2local**: Needed to get a list of CE,UID,PID matching with a dg_jobid. It will be used on the HLR side to control a job that has overrun the fund reserved.
- **local2dg_jobid:** Needed to get the dg_jobid of a job knowing the local Unix PID. It will be used on the CE side to trace back to the HLR of the user who submitted the job.
- **sensor_x:** Gives punctual and integral information about the resource x (e.g. CPU time) for a job with a given PID, if the job exits, it returns the last available values and the exit status.

In our working scheme, a daemon called **JobHunter** waits for new grid jobs, once a job has been started by the gatekeeper the daemon gets the PID of the process and gives it to a process, called **JobMonitor**, that, while the job runs, runs the sensors **sensor_x** and collects the needed information.

The JobMonitor then cyclically communicates the job resources usage to the user HLR, to do so it uses the **local2dg_jobid** tool to discover the dg_jobid of the job and then the user HLR address.

Once the job has finished its cost can be computed and paid to the resource HLR.

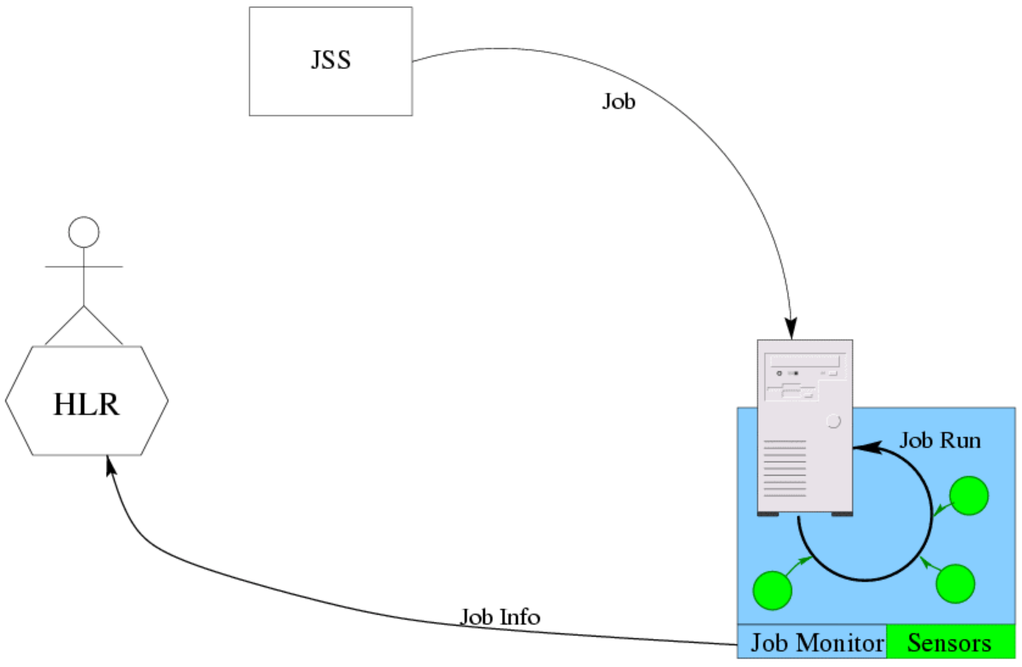

Figure 3 JobMonitor and sensors scheme

On the user HLR side, cyclically the total cost of a running job is computed and, if it exceeds the maximum foreseen for that job, it can be halted by sending an appropriate signal to the **JobMonitor** of every part of the job on the many CEs that are actually executing it. This can be accomplished by getting the CE,UID,PID of every process of that job by the **dg_jobid2local** tool.

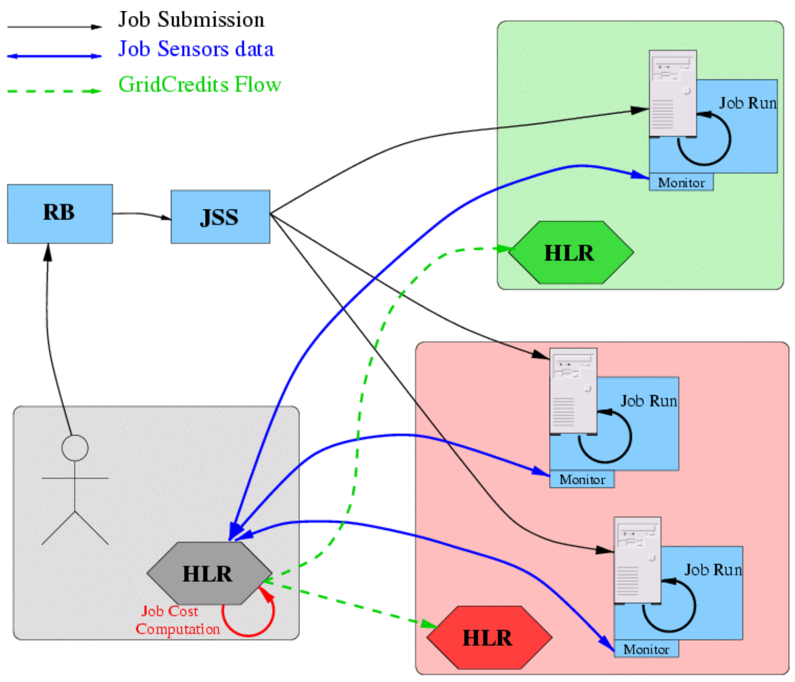

Figure 4 Job Monitor and cost computation data flow

This behaviour of the system is needed to cope with jobs that, in principle, can be partitioned or checkpointed and then moved to another CE.