



REPORT ON IMPLEMENTATION OF JRA1 PROTOTYPES

DESCRIPTION OF DJRA1.2 AND 1.3 DELIVERABLES

Document Filename:	BG-DJRA1.2-3-v0.1
Activity:	JRA1
Partner(s):	PSNC, KTH
Lead Partner:	PSNC
Document classification:	PUBLIC

Abstract: The report describes the current state of the products developed in research activity (JRA1). The first part of the document describes Service Level Agreement based on Tycoon integrated with gLite middleware, including brief architecture, administrator guide and test installation (prototype of D.JRA1.2). The second part of the document describes account management system based on PSNC Virtual User System integrated with gLite middleware, including general architecture, administrator guide and installation procedures (prototype of D.JRA1.3).





REPORT ON IMPLEMENTATION OF JRA1 PROTOTYPES
Description of DJRA1.2 and 1.3 deliverables

Document review and moderation

	Name	Partner	Date	Signature
Released for moderation to				
Approved for delivery by				

Document Log

Version	Date	Summary of changes	Author
0.1	28/05/2007	Initial version	Michal Jankowski Norbert Meyer Amir H. Payberah Lars Rasmusson Niklas Wirström
0.5	08/06/2007	Tycoon test installation described	Michal Jankowski Amir H. Payberah
1.0	11/06/2007	Update of the report	Michal Jankowski Amir H. Payberah Norbert Meyer
1.1	18/06/2007	Conclusions added. Glossary revised.	Michal Jankowski Norbert Meyer



Contents

GLOSSARY.....	4
1. EXECUTIVE SUMMARY.....	6
2. DJRA1.2 – PROTOTYPE OF SLA QOS.....	7
2.1 SYSTEM DESCRIPTION.....	7
2.1.1 <i>Tycoon</i>	7
2.1.2 <i>gLite</i>	8
2.1.3 <i>The Tycoon-gLite Integration</i>	9
2.2 IMPLEMENTATION.....	11
2.2.1 <i>General Tycoon Account Creation</i>	11
2.2.2 <i>Virtual Cluster Creation</i>	13
2.3 INSTALLATION AND CONFIGURATION.....	16
2.3.1 <i>Installing Tycoon-gLite platform</i>	16
2.3.2 <i>Xen</i>	16
2.3.3 <i>Tycoon-glite</i>	20
2.3.4 <i>Creating gLite File system Image</i>	25
2.4 FUTURE WORK.....	27
2.5 TEST INSTALLATION.....	28
3. DJRA1.3 - PROTOTYPE OF ACCOUNT MANAGEMENT SYSTEM.....	29
3.1 SYSTEM DESCRIPTION.....	29
3.2 IMPLEMENTATION.....	30
3.2.1 <i>Assumptions</i>	30
3.2.2 <i>Architecture</i>	30
3.2.3 <i>VOMS Attributes</i>	31
3.2.4 <i>Authorization</i>	31
3.2.5 <i>VUS Database</i>	31
3.2.6 <i>Use Scenario</i>	32
3.3 INSTALLATION AND CONFIGURATION.....	33
3.3.1 <i>Installation and Initial Configuration</i>	33
3.3.2 <i>Initial Configuration</i>	33
3.3.3 <i>Advanced Configuration</i>	34
3.4 FUTURE WORK.....	35
3.5 TEST INSTALLATION.....	35
4. CONCLUSIONS.....	36
REFERENCES.....	37



GLOSSARY

CE	Computing Element
DGAS	DataGrid Accounting System
DHCP	Dynamic Host Configuration Protocol
DN	Distinguished Name
DNS	Domain Name Service
EGEE	Enabling Grids for E-science
FQAN	Fully Qualified Attribute Name
Gianduia	Gianduia Is A Nice Distributed Usage-metering Infrastructure for Accounting. It is a daemon –module of DGAS, installed on a CE in order to collect the usage records of the executed user jobs and send them to the DGAS HLR service.
GID	Group Identifier
GRAM	Grid Resource Allocation and Management
GSI	Grid Security Infrastructure
HLR	Home Location Registers
IP	Internet Protocol. Also IP address.
JDL	Job Description Language
LCAS	Local Center Authorization Service
LCMAPS	Local Credential MAPPING Service
LRMS	Local Resource Management System
LVM	Logical Volume Management
NAT	Network Address Translation
QoS	Quality of Service
R-GMA	Relational Grid Monitoring Architecture
RPM	Red Hat Package Manager
RSL	Resource Specification Language
SSH	Secure Shell
SLA	Service Level Agreement
SLS	Service Location Server
Torque	An open source local resource management system providing control over batch jobs and distributed compute nodes.
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol



REPORT ON IMPLEMENTATION OF JRA1 PROTOTYPES
Description of DJRA1.2 and 1.3 deliverables

UID	User Identifier
VM	Virtual Machine
VO	Virtual Organization
VOMS	Virtual Organization Membership Service
VPN	Virtual Private Network
VUS	Virtual User System
WM	Workload Manager
WMS	Workload Management System
WN	Worker Node



1. EXECUTIVE SUMMARY

The Joint Research Activity of BalticGrid project (JRA1) is focusing on SLA security and enforcement, account management and accounting and will deploy these services in the BG in co-operation with SA1 and with support of SA2. Specifically we investigate, prototype and suggest mechanisms for ensuring that SLAs are established and enforced in a standard and secure way, design, develop and deploy, in cooperation with the Center Operations activity, a non-intrusive, standards based user account management system which can be integrated into a Grid accounting system, carry out research, prototype and in collaboration with other activities, deploy SLAs in the BG.

The objectives of JRA1 are following [5]:

- to investigate, prototype and implement mechanisms for ensuring that SLAs are established and enforced in a standard and secure way,
- to research, prototype and deploy SLAs in the BG,
- to design, develop and deploy, a non-intrusive, standards based user account management system.

In order to reach the above objectives, prototypes of SLA and user account management systems were designed and implemented. BalticGrid project focuses on extending EGEE infrastructure to the Baltic states, so that using EGEE software -gLite is assumed. Another assumption is reusing existing solutions as much as possible. To fulfil these assumptions, the proposed systems are based on existing tools, previously developed by the project partners and gLite software. This document describes these systems.

User-level Service Level Agreements mean the possibility for users to acquire more (or less) resources in a well defined manner [5]. The resource consumption will be accounted for, to prevent some users from asking for so much resources that other users are starved. A market-based resource allocation system makes efficient use of its resources since it encourage users not only to choose resources with currently low demand, but also to choose time periods when demands are low. It also allocates more resources to more important tasks, if tasks are funded according to importance. Thus, the service level of a task or group of tasks is ultimately determined by its level of funding, relative to other tasks [2]. One of such a systems is Tycoon. We use the Tycoon market-based Utility Computing infrastructure to create and manage a SLA-capable execution environment hosting unmodified gLite Computing Elements. Section 2 describes system that integrates the gLite and Tycoon platforms.

The main aim of user management system is controlled, secure access to grid resources. The scope of features offered by such a system may vary significantly, depending on the needs of users and administrators. The user management system that is part of gLite had to be enhanced to fulfil, well defined BalticGrid requirements. It could not interfere in the sites that are not going to use the system. The special focus was put on the proper level of security, job isolation, automation of the administrative work and compliance with standards. Another objective was support for accounting and easy integration with the accounting system to be deployed in BalticGrid. The resulting system is based on Virtual User System developed by PSNC, integrated with gLite. Section 3 describes this integrated system.

The prototypes were installed and tested in KTH (Tycoon) and PSNC (VUS and accounting system – DGAS) under local testbeds. The next step is to increase the pilot installation in EENET, IMCS UL and VU, so that the pilot installation will consist of 5 sites. The final stage planned to be done until the end of 2007 is to deploy all services in SA1 infrastructure environment.



2. DJRA1.2 – PROTOTYPE OF SLA QOS

2.1 SYSTEM DESCRIPTION

Here we describe the Tycoon-gLite system. This is done by first briefly describing the current implementation of the Tycoon system, and then (also briefly) describe the functionality of the gLite system. Last in this section we describe the integrated system.

2.1.1 Tycoon

A detailed description of the Tycoon system can be found in [1].

Tycoon is a market-based system that manages computing resources like CPU, RAM, disk, network bandwidth, etc., for users that compete for the resources. The aim of Tycoon is to provide users with an intuitive and extremely simple way to specify resource demand, namely through a single variable, the "spending-rate".

The Tycoon system consists of a Bank, Auctioneers, a Service Location Server and Agents. Auctioneers are responsible for allocating and to do accounting for resources on the host where they are installed. The Auctioneers publish themselves in an SLS. Agents, which each acts on behalf of a resource user, queries the SLS for information about prices and location of resources. After selecting the economically most efficient set of Auctioneers, an Agent uses the Bank to transfer money from its user's bank account to that of the Auctioneer.

Tycoon enables users to create private Virtual Machines, and assigns resource shares to the VM's, proportionally to the spending-rate. Tycoon also contains a secure banking infrastructure that provides authenticated money transfer between the users and the resource providers.

The current implementation of Tycoon uses Xen as the virtualization layer. A detailed description of Xen can be found in [3]. Xen can dynamically create and reallocate resources between virtual machines. One of the VMs that run on top of Xen is a *privileged domain* that has access to the physical devices (disk, network cards, etc.). It is called *domain0*, and is the domain in which the Tycoon Auctioneer runs. Many users run concurrently on the same physical machine, isolated from each other, in their individual virtual machines.

The privileged domain acts as a NAT firewall for the user VMs. Each Tycoon VM has at least one port which is reachable from the outside. It is forwarded with Linux `iptables` to the VM's SSH port.

Users can bid for additional TCP ports from Domain0's host address. Such externally visible ports make incoming external communication possible, and avoids the overhead of setting up a VPN.

It is also possible for the Auctioneer to possess a pool of IP addresses for which the users may bid. In this way a VM can be assigned a public IP address.

Figure 1 shows the steps involved in a Tycoon host account creation: 1) An Tycoon Agent queries an SLS for information about Auctioneers and prices. 2) The SLS responds with this information, and the Agent selects a set of Auctioneers. For each Auctioneer in the set, the Agent then does the following: 3) The Agent contacts the Bank, requesting a money transfer from its user's account to that of the Auctioneer. 4) The Bank sends back a signed receipt to the Agent. 5) The Agent forwards the receipt to the Auctioneer which then creates an Tycoon account and a VM. 6) When the VM has booted, the user can log in to it and start to use it.

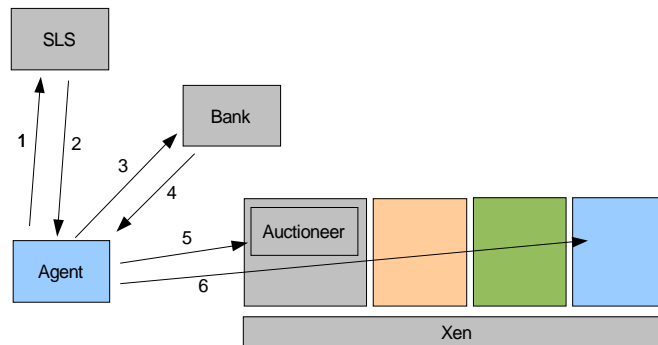


Fig. 1 Tycoon host account creation.

2.1.2 gLite

The components of the gLite system can be divided into two categories: Site level components and Grid level components.

Here and in the rest of this report, we refer to a *site* as subsystem of gLite which provides job and data management systems, and information and monitoring services regarding these. The job management system is comprised of a CE and a number of WNs. The CE acts as an interface to the Grid level, and is responsible for delegating jobs to the WNs which performs the actual work. Each CE possesses a *host certificate* for authentication.

The data management system is responsible for storing data, and to map globally unique identifiers to local file names. The information service is responsible for publishing information of the state of, for example, CEs or jobs.

On the Grid level, components responsible for Grid security, information services, and workload and data management systems are found. The information service on this level is responsible for binding information publishers at the site level and information consumers such as a user querying the status of a job. The data management system keeps track of where particular files are stored, and the WMS is responsible for matching jobs with available CEs.

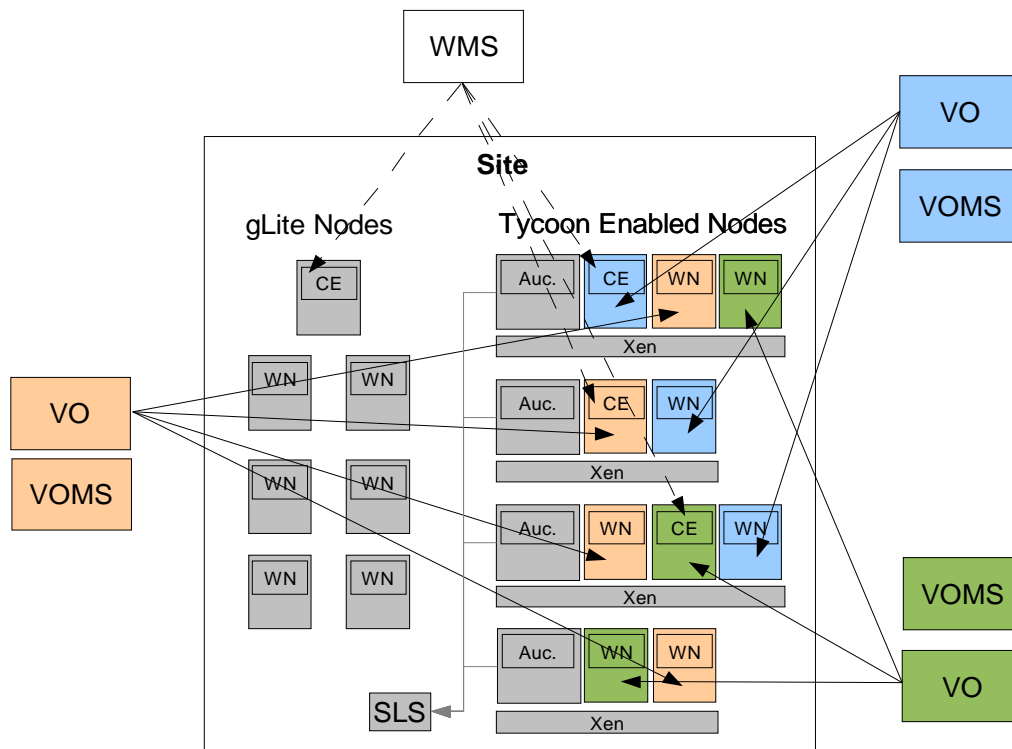


Fig. 2 Three different VOs (orange, blue and green) and their virtual clusters on a specific site

A VOMS server constitute the Grid security. A VOMS server acts as a repository for information of user authorization. Components such as CEs and WMSs, use a VOMS server's SOAP interface to receive such information.

Further information of the gLite system can be found in [4].

2.1.3 The Tycoon-gLite Integration

The BalticGrid is a production environment as well as a research environment. To avoid disturbing the users of the BalticGrid unnecessarily, we provide a heterogeneous gLite system, in which some compute nodes can run the unmodified gLite software, and other nodes can run gLite-prepared Tycoon Auctioneers.

The gLite WMS is used for submitting jobs and Tycoon SLSs are used for discovering resources for creation of virtual clusters. Each site runs an SLS for each LAN on the site that contains gLite-prepared Tycoon Auctioneers. All Auctioneers on the same LAN registers in the same SLS.

VO managers can request the Auctioneers to create several VMs that together constitute a virtual gLite compute cluster. In each virtual cluster there is one CE and an arbitrary number of WNs. Only members of the VO owning a virtual cluster are able to submit jobs to it.

The CE belonging to virtual clusters register with the already existing BalticGrid infrastructure, and look just like regular CEs to the BalticGrid users. A user that submits a job to the BalticGrid can either let the gLite WMS pick a suitable computing element for its job, or specify a CE belonging to one of its VO's virtual clusters.

The virtual clusters use TORQUE as the Local Resource Management System.

The system is also depicted in Figure 2. The figure shows a site with both ordinary gLite nodes and gLite-prepared Tycoon Auctioneer nodes, all using the same WMS. All auctioneers in the picture are



on the same LAN, and registers therefore in the same SLS. Each VO has a VOMS server which its CE uses to retrieve VO-specific information.

2.1.3.1 Example Scenario

The following is a description of an example scenario, where a VO manager creates a virtual cluster on a Tycoon enabled gLite site. For further information about the commands named below, see Section 3.2

1. The VO manager has previously been allocated a budget on a Tycoon-managed bank account. The VO manager also has a list of SLSs, each corresponding to a LAN where gLite-prepared Tycoon Auctioneers are installed.

2. The VO manager issues the following command to get information from the different SLSs:

```
multi_wn_agent list <sls_list> <budget> <deadline> <max_nodes>
```

Here, <sls_list> is the list of SLSs, <budget> is the amount of money to be spent, <deadline> is the amount of seconds over which the budget is to be spent, and <max_nodes> is the maximal number of nodes wanted. This outputs a list of Auctioneers for each SLS in <sls_list>.

3. To create a CE, the VO manager chooses an Auctioneer from one of the lists output by the multi_wn_agent list command and issues the following command:

```
ce_agent create <ce_name> <auctioneer> <budget> <ce_ip>  
<ce_netmask>
```

4. Here, <ce_name> is the name of the VM to be created, <auctioneer> is the IP address to the Auctioneer where the VM is to be created, <budget> is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto. <ce_ip> and <ce_netmask> are the VMs public IP address and netmask, respectively.

5. The VO manager sets the environmental variable CE_SLS to point to the SLS corresponding to the chosen Auctioneer.

6. To create worker nodes, the VO manager issues the following command:

```
multi_wn_agent create <ce_name> <wn_pref> <budget> <deadline>  
<max_nodes>
```

Here, <wn_pref> is a tag that precedes the name of each WN to be created. This command outputs a list of Auctioneer - VM name pairs, which can be used when rebalancing the bids.

7. Users belonging to the specific VO can at this point start submitting jobs to the virtual cluster. As usual, the user first creates a VOMS Proxy with the following command:

```
voms-proxy-init -voms <VOMS_SERVER>
```

where <VOMS_SERVER> is a VOMS server that holds information of the VO to which the virtual cluster and the user belongs. Now, the user can submit a job by issuing:

```
glite-job-submit -r <CE_RESOURCE> <JDL>
```

where <CE_RESOURCE> specifies the CE's IP address, its Gatekeeper port number and the desired service; and <JDL> is a normal Job Description Language file.

8. To rebalance the bids over the set of existing WNs, the VO manager issues the following command:



```
multi_wn_agent rebalance <rate> <info_list>
```

where <rate> is the spending rate and <info_list> is the list output in step 5.

2.1.3.2 Funding of Resources

The first resource allocation is the allocation of money to the Virtual Organizations. The amount of money allocated can be different for each VO, and is determined by a human committee.

The money is used by a VO to purchase resources from the Tycoon-managed machines, which creates a virtual cluster for the virtual organization. The amount of resources allocated to the virtual cluster is determined by the current demand from the other virtual organizations in the BalticGrid. When the users in the VO need more resources, they will ask the VO manager to increase the spending rate. The rate will be decreased when the demand is low again.

A VO manager uses a software agent that gets as input a spending rate, measured in EUR/second. The agent is used to select optimal Auctioneers from a given SLS, create and configure CE and WN nodes, and to rebalance the VO's total funds over the already created VMs, on certain time intervals. The gLite-Tycoon system itself does not provide information of on which WN a specific job is being run. Therefore all jobs executed on the same virtual cluster have the same probability to benefit from an incrementation of the spending rate. This means that a low priority job actually may have greater computing power than a high priority job running on the same virtual cluster. Such information can, however, be manually extracted from the system, in which case it can be used to manually allocate more or less money for a specific WN.

The VO manager can also use the Agent to, at any time, create new nodes or delete existing ones. When the Agent selects Auctioneers or rebalances the total funds over an already existing set of machines, it is done so that the marginal price per selected resource is the same (and sum up to the spending rate). In the case of selection, also, all the other machines are more expensive. This is referred to as the *best response* algorithm [1].

In the current implementation the number of nodes to run on is chosen as the number that maximizes the amount of resources per EUR spent, but in some cases, the VO may want to enforce a bigger or smaller number of nodes in the virtual cluster. This can be done by manually creating or deleting nodes.

2.2 IMPLEMENTATION

Here we explain how the system is implemented and plugged in to the existing gLite infrastructure, and the way CEs and WNs are created to form a virtual cluster.

2.2.1 General Tycoon Account Creation

When a Tycoon Agent creates an account at an Auctioneer, it first makes a bank transfer from its user's bank account to that of the owner of the Auctioneer. The bank returns a signed receipt to the Agent. The agent sends a *create request* to the Auctioneer, attaching the receipt and options regarding the configuration of the user's Tycoon account. These options include user name, public keys used for future log-ins, etc. It is also possible to specify the path to a file system to be used by the new VM.

The Auctioneer now creates a new Tycoon user account, and requests Xen to create a new VM. The user's public key is made an *authorized key* of the root account of the VM.

Xen assigns a locally unique number to each VM. We refer to this number as `domain_ID`.



2.2.1.1 Network Configuration

When a new VM is created, Xen creates two virtual network interfaces. One in the privileged domain, and one in the new VM. Xen then decides MAC addresses for these interfaces.

The creation script can be called with an argument specifying a public IP address for the VM's virtual interface. If no such address is specified, the interface will be assigned an address on the form *10.X.Y.Z*, where the four leftmost bits of *X* are specific to the particular physical host. The remaining bit string (i.e. the four rightmost bits of *X*, and the bits of *Y* and *Z*) is the binary representation of *domain_ID*2*.

The creation script assumes that the privileged domain has access to a local DNS server in order to add and remove entries as VMs are created and deleted.

It is also assumed that the privileged domain runs a DHCP server for its VMs.

1. The back-end interface is assigned an IP address according to the procedure described above.
2. If a public IP address was specified in the arguments, that address is assigned to the VM's virtual network interface. Otherwise it is assigned an address on the form *10.X.Y.Z* as described above.
3. If no public address was specified, an iptables rule specifying to *masquerade* all IP packets with source address matching that of the VM's network interface, and that are not destined to hosts with an IP address matching *10.0.0.0/8*.

If a public address was specified, an iptables rule specifying to *accept* all IP packages destined to this address. Also, the privileged domain is set up to do proxy-ARPing for the VM.

4. An entry for the VM's host name is created in the `/etc/dhcpd.conf` file in the privileged domain. This overwrites any already existing entry with that name.
5. The DHCP Server in the privileged domain is reloaded with the new configuration.
6. The local DNS is configured to handle the new host. This is done using the `nsupdate` command.

2.2.1.2 Computing Elements and IP Addresses

In Tycoon, each user domain is NATed by default. The authentication in gLite unfortunately uses reverse DNS for host authentication, which is problematic for NATed compute clusters. The reason is that the host name is entered in the certificate subject, and the communicating partner verifies the host name of a caller with reverse DNS. So all CEs behind a NAT will appear to have the same host name. Another problem with using NAT is that a large number of ports have to be forwarded to each CE. If some ports are not reconfigurable, they will not allow two virtual CEs share the external IP address of the same physical machine.

For this reason, CEs have public IP addresses and host names published in a DNS. Because of this, each CE must have a host certificate. How these are provided can be implemented in the following ways:

1. The site is responsible for providing the host certificates to the virtual CEs. Since a host certificate with a Distinguished Name of `ce.mysite.se`, can be used by hosts named `ce-1.mysite.se`, `ce-2.mysite.se`, etc., it is sufficient to have a single certificate that is used for all CEs on the site.

Since the same certificates then are used by multiple CEs belonging to different VOs, the certificate's private key must not be readable to the VO manager. Therefore, only the site administrator, and not the VO manager can be allowed to log in as root on the CE. In this way the virtual CEs have the same level of credential as an ordinary CE belonging to the same site.



2. The VO is responsible for providing the host certificates for its CE. In this way, the VO manager can be allowed to administrate the CE. However, this means that each VO needs to, either, have a separate host certificate for each site in the Grid and let the site configure the proper DNS entries (both direct and reversed), or to choose a certificate with a DN from its own network domain, and configure a DNS to point direct DNS lookups of that name to the IP address (belonging to the site) of the CE. In the latter case the site must configure a DNS to point reversed DNS lookups to the host name.

We have chosen the first strategy for providing host certificates for CEs. However, this means that the Tycoon Auctioneer software needs to be slightly modified.

2.2.1.3 Worker Nodes and IP Addresses

WNS do not need public addresses for communication between each other or with a CE. Since all nodes reside on the same LAN, the following strategy can be used:

When the VM is created, it is equipped with a virtual network interface. In the privileged domain, another virtual network interface which is connected to that of the VM, is created.

All virtual interfaces on a particular physical machine, are assigned IP addresses from a predefined range, specific to that machine. In order to allow VMs on one physical machine to communicate with VMs on another, each privileged domain has a routing entry for each of the predefined address ranges specifying the associated privileged domain as the gateway.

2.2.1.4 File System Configuration

Each VM has its own file system. We use Logical Volume Management to create copy-on-write *snapshots* of a pre-installed file system. An LVM snapshot can be created within seconds, which makes fast VM creation possible.

Before the VM is booted, the snapshot is mounted and the user's public SSH key is copied to it.

2.2.2 Virtual Cluster Creation

To create a virtual cluster, the VO manager creates a VM containing a CE and several others containing WNs. When the VMs are booted they use a file system where the gLite software has been preinstalled, but not yet configured. Configuration of the VMs include specifying the CE's IP address to the WNs and vice versa, and settings regarding restricting the virtual cluster to only be used by one VO. This will be done automatically by a software agent, without the interaction of the VO managers or BalticGrid administrators. Below, we give a more detailed explanation of how the virtual clusters are created. The scripts and commands in the following sections are prototypes.

2.2.2.1 Selecting a Site for Virtual Cluster Creation

A virtual cluster can be created on any site which has installed gLite-prepared Tycoon Auctioneers and the other requirements found in Section 4.

It is assumed that each VO manager possesses a list of SLSs, where each SLS corresponds to a LAN with gLite-prepared Tycoon Auctioneers. The VO manager can then use the following command to list the `<max_nodes>` most economical machines for each SLS in the list:

```
multi_wn_agent list <sls_list> <budget> <deadline> <max_nodes>
```

Here, `<sls_list>`, is the list of SLSs, `<budget>` is the amount of money to be spent, `<deadline>` is the amount of seconds over which the budget is to be spent, and `<max_nodes>` is the maximal number of nodes wanted. The command runs the *best response* algorithm for each SLS in



<sls_list>, and outputs a list of Auctioneers. The list is grouped by LAN (corresponding to an SLS), and specifies the amount of money that should be spent on each Auctioneer and the resulting utility value of doing so, given that the VO manager chooses to create the virtual cluster on that particular LAN.

By comparing the utility values of the Auctioneers of different LANs, the VO manager can select an economically good set of machines for creation of a virtual cluster. When this choice has been made, the VO manager first picks a machine on which he/she creates a CE, and then WNs can be created.

2.2.2.2 CE Creation

To create a CE on a specific Auctioneer the following command can be used:

```
ce_agent create <ce_name> <auctioneer> <budget> <ce_ip> <ce_netmask>
```

Here, <ce_name> is the host name of the VM to be created, <auctioneer> is the IP address or host name of the Auctioneer where the VM is to be created, <budget> is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto. <ce_ip> and <ce_netmask> are the VMs public IP address and netmask, respectively. These arguments should in future versions of the system be decided by the site instead of the VO manager.

The `ce_agent` also needs some additional parameters that can either be set as environmental variables or edited directly in the `ce_agent` script.

The Auctioneer uses a special creation script to set up the VM as a CE. Instead of making the VM's root account accessible to the VO manager, the script creates a special user account (*voadmin*) on the VM for which the VO manager's public key is made an *authorized key*. In order to let a VO manager configure the CE, and add or delete WN's to the LRMS, this user is made able to execute special scripts with root permissions, using the `sudo` command. In this way the host certificate's private key can remain private to the site administrator.

When the VM's file system has been set up, the VM is booted and it can be configured. The Agent then logs in to the machine (as *voadmin*) and runs a configuration script with root privileges. This script uses the gLite configuration tool and the gLite site configuration file to configure the CE. Values in the site configuration file has partly been specified by the site administrator when setting up the Auctioneer, but some values are VO specific and must be set up at this point: A VOMS server needs to be specified in the `VO_<VONAME>_VOMS_SERVERS` variable in the site configuration file. Here, <VONAME> is the name of the VO. Additional VO specific variables that need to be specified includes user- and group settings for local UNIX accounts.

All VO specific settings will be read from files by the VO's Agent and handed over to the configuration script.

The gLite configuration tool uses the `source` command to read configuration parameters from a file. No special check is made which results in that any command in the file may be executed. Therefore the file supplied by the VO manager cannot simply be appended to the site manager's configuration file since it may contain malicious code. We solve this in the following way:

1. When a VO manager creates a CE, he/she provides a file containing the VO specific configuration parameters.
2. A script extracts lines starting with a recognized variable name and rewrites it to have the following format:

```
<var_name>=$(echo '<uu_encoded_value>' | uudecode )
```

Here, <var_name> is the variable name <uu_encoded_value> is the 64 bit uu encoded version of the value specified by the VO manager.



2.2.2.3 WN Creation

To create a WN the following command can be used:

```
wn_agent create <wn_name> <auctioneer> <ce_name> <budget>
```

Here, <wn_name> is the name of the WN, <auctioneer> is the IP address of the Auctioneer where the VM is to be created, <ce_name> is the name of the CE that should be configured to use the WN, and <budget> is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto.

The Auctioneer creates a VM in the normal Tycoon manner and boots it. When the VM is booted, the agent copies the site configuration file from the CE to it, and logs in to the VM as root and use the gLite configuration tool to configure it as a WN. The agent also adds the new WN to the CE's WN list. However, the LRMS on the CE needs to be reconfigured before the new settings can be used. This is done with the following command:

```
wn_agent update <ce_name>.
```

To set up multiple WNs, it is convenient to use the following command:

```
multi_wn_agent create <ce_name> <wn_pref> <budget> <deadline>  
<max_nodes>
```

This command runs the `wn_agent create` command for the most economical set of nodes on the LAN where the CE is created. <wn_pref> is a tag that is to precede the name of each WN to be created, <deadline> is the length of the period over which the budget should be spent, and <max_nodes> is the maximum number of nodes that are to be created. When WNs have been created and configured the script automatically runs the `wn_agent update <ce_name>` command.

Using this command can cause a WN to be created on the same physical machine as the CE of the same virtual cluster. This would result in that the VO competes with itself for the same resources.

2.2.2.4 Funding a Virtual Cluster

Funding nodes of a virtual cluster is done in the usual Tycoon manner. That is, the Agent transfers money from its user's account to the account of the owner of the Auctioneer. The bank sends back a signed receipt to the Agent. The Agent sends the receipt along with a *funding request* to the Auctioneer, specifying which account to be funded. The Auctioneer adjusts the user's local account balance according to the funding amount.

2.2.2.5 Rebalancing Bids of a Virtual Cluster

To rebalance the bids, the following command can be used:

```
multi_wn_agent rebalance <rate> <wn_info>
```

Here, <rate> is the spending rate (i.e. budget / deadline) and <wn_info> is the list of Auctioneer - VM name pairs that was output by the `multi_wn_agent create` command. The command calculates the total amount of money remaining on the accounts of the VMs specified in the <wn_info> list, and rebalance the money over these accounts so that the most economical configuration is obtained.



2.3 INSTALLATION AND CONFIGURATION

This section describes how to install the gLite-Tycoon system on a site. The installation regards the installation of the virtualization and Tycoon software, the preparation of a file system image to be used by the VMs acting as CEs or WNs.

2.3.1 Installing Tycoon-gLite platform

To enable a site to create virtual clusters and CEs, the Tycoon software must be installed on a set of the physical nodes. Each site also needs a DNS at its disposal, and an SLS for each LAN containing gLite-prepared Tycoon Auctioneers.

The following must be done on each of the physical cluster nodes that will host virtual cluster nodes:

1. Install the Xen virtualization software.
2. Install a Fedora Core root file system.
3. Install NTP.
4. Install the gLite-prepared Tycoon Auctioneer software.
5. Set up routing tables.
6. Install keys for permission to modify the local DNS.
7. Install a DHCP server.
8. Install Scientific Linux file system images for use by the Tycoon-managed CEs and WNs. These images should have the gLite software pre-installed so that only the gLite configuration step is necessary when creating new virtual cluster nodes.

In the remaining part of this section it will be explained how to install Xen, Tycoon-gLite and also how to prepare Tycoon-gLite file system image.

2.3.2 Xen

To enable the machine to support Tycoon Auctioneer, Xen should be installed. In this section installing and working with Xen is explained.

2.3.2.1 Installing Xen

Installing Xen can be done in two different ways: from source code or from its RPM. To install Xen from its RPM, the following command can be used.

```
# yum install xen kernel-xen0 kernel-xenU
```

This command installs the required packages. *xen* is the Xen OS kernel; *kernel-xen0* is the Xen-enabled host system kernel (domain 0) and *kernel-xenU* is the Xen guest system kernel.

After installing these packages, grub configuration should be changed to support the new installed Xen kernel. The grub.conf should be consist of some lines as follow which defines the Xen and Xen0 kernel path:

```
title Fedora Core (2.6.19-1.2288.fc5xen0)
    root (hd0,0)
    kernel /boot/xen.gz-2.6.19-1.2288.fc5
    module /boot/vmlinuz-2.6.19-1.2288.fc5xen0 ro root=LABEL=/1
rhgb quiet selinux=0
```



```
module /boot/initrd-2.6.19-1.2288.fc5xen0.img
```

Once the system is being rebooted with the xen0 GRUB option, it will start with a Xen-enabled kernel. After that, Xen service should be enabled. Enabling Xen can be checked with following command:

```
# /usr/sbin/xm list
```

Now Xen is enabled and can manage some guest OS concurrently.

2.3.2.2 Creating Xen file system image

To create Xen file system image, at first, the required OS should be installed on one partition. But if an empty partition is not available, the best solution is to use QEMU. QEMU is a generic and open source machine emulator and virtualizer. With QEMU it is possible to install the OS on one image file.

To install the required OS with QEMU, at first one image file should be created:

```
# qemu-img create qemu-image.img 1G
```

This command creates an image file with size of 1G. After creating this image file, it is possible to install the OS on it.

```
# qemu -m 128 -hda qemu-image.img -cdrom /dev/cdrom -boot d
```

This command installs the OS from cdrom on the created image file. It considers 128M for RAM.

The next step after installing the OS is converting its format from QEMU to Xen image format. Following shows the steps of conversion:

1. Make a couple directories to mount the images.

```
# mkdir /mnt/loop1; mkdir /mnt/loop2
```

2. Mount the QEMU image to loop1 (in the following command should be calculated).

```
# mount -o loop,offset=##### qemu-image.img /mnt/loop1
```

To calculate ##### at first use fdisk command to find out the start sector of root file system on image file and then multiple it at 512. For example:

```
# fdisk -lu qemu-image.img
```

You must set cylinders.

You can do this from the extra functions menu.

Disk qemu-image.img: 0 MB, 0 bytes

255 heads, 63 sectors/track, 0 cylinders, total 0 sectors

Units = sectors of 1 * 512 = 512 bytes

Device	Boot	Start	End
Blocks	Id	System	
qemu-image.img1	63	257039	128488+ 82
Linux swap / Solaris			
qemu-image.img2	*	257040	2088449 915705 83
Linux			



Partition 2 has different physical/logical beginnings (non-Linux?):

```
phys=(127, 16, 1) logical=(16, 0, 1)
```

```
# mount -o loop,offset=131604480 qemu-image.img /mnt/loop1
```

In this sample offset is equal $512 * 257040 = 131604480$, so the mount command should be:

```
# mount -o loop,offset= 131604480 qemu-image.img /mnt/loop1
```

3. Create a new sparse image file and format it (The seek number is how many MB to make the file.)

```
# dd if=/dev/zero of=xen-image.img bs=1M count=1 seek=1024
```

```
# mke2fs -F -j xen-image.img
```

4. Mount the spare image.

```
# mount -o loop xen-image.img /mnt/loop2
```

5. Copy everything over from QEME image to Xen image. 1.

```
# cp -ra /mnt/loop1 /mnt/loop2
```

6. Create essential devices on Xen image. 1.

```
# for i in console null zero; \  
do \  
/sbin/MAKEDEV -d /mnt/loop2/dev -x $i; \  
done
```

7. Change Xen image fstab. 1.

```
/dev/sda1    /                ext3          defaults      1 1  
none        /dev/pts         devpts       gid=5,mode=620 0 0  
none        /dev/shm         tmpfs        defaults      0 0  
none        /proc            proc         defaults      0 0  
none        /sys             sysfs        defaults      0 0
```

8. Disable TLS on Xen image.

```
# echo "hwcap 0 nouseg" >> /mnt/loop2/etc/ld.so.conf
```

9. The last step is to create xenU kernel for Xen image. It can be done with yum as said before (yum install kernel-xenU) or can be done from source file. Using source file has advantage of configuring kernel according to the situation. To compile the Xen from source after downloading it, do the following steps: 1.

```
# cd xen-<xen-version>
```

```
# make world
```

```
# make install
```

```
# ./install.sh
```



```
# cd linux-<kernel-version>-xenU
# make menuconfig
# make
# make modules_install
# cp vmlinuz /boot/vmlinuz-<kernel-version>-xenU
# cp -ra /lib/modules/<kernel-version> /mnt/loop2/lib/modules/
```

10. Finally, just in case of using Fedora RPM's to install, it is required to add the following line to the `/etc/modprobe.conf` file in Xen image file1.

```
alias eth0 xennet
```

To make changing image kernel easily, it's better to put kernel modules in separate image file and then mount it separately, instead of put it in Xen image file (But this step is optional).

```
# dd if=/dev/zero of=modules bs=1M count=100
# mke2fs -F -j modules
# mount modules /mnt/modules
# cp -ax /lib/modules/<linux xenU version> /mnt/modules
# umount /mnt/modules
```

add this line to Xen image `etc/fstab`

```
/dev/sda2 /lib/modules ext2 defaults 0 1
```

2.3.2.3 Configuring and working with Xen

Tycoon-gLite auctioneer manages the guest OS without interference of others, but it would be useful to know how Xen manages its guest OSes.

Xen creates a configuration file for each guest OS under `/etc/xen`. The sample configuration file for Scientific Linux 3 can be as follow:

```
kernel = "/boot/vmlinuz-<kernel-version>-xenU
memory = 256
extra = "fastboot ro selinux=0"
ramdisk = "/boot/initrd-<kernel-version>-xenU.img"
disk = ['file:/root/xen-image.img,sda1,w', 'file:/root/swap,hda5,w']
hostname = "sl"
name = "sl"
root = "/dev/sda1 ro"
```

In this file, `kernel` refers to Xen enabled kernel of guest OS (xenU). The other important parameter is `disk` which refers to the file system image. This image can be a image file or an installed OS on disk. If modules are installed on separate image file, the disk configuration can be like this:

```
disk = ['file:/root/xen-image.img,sda1,w',
'file:/root/modules,sd2,r']
```

If the guest OS was installed on disk the disk parameter should be change as follow:



```
disk = ['phy:/dev/hda5,sda1,w', 'phy:/dev/hda6,sda2,w']
```

On some machines the root file system check will bring domain 0 into single user mode. The `fastboot` command line option will make SL skip this check.

The `root` parameter also points to the virtual partition which is mounted as root on machine (it is defined in `/etc/fstab` in file system image).

The only thing to test the guest OS is to boot it. This can be done by executing following command:

```
xm create -c /etc/xen/<your-config-file>
```

2.3.3 Tycoon-glite

In this section the Tycoon-gLite structure and also installing and working with it is explained.

2.3.3.1 Tycoon-gLite source code

Tycoon consists of four types of components:

- **Clients:** Client programs (also called agents) help users set up and manage accounts, find resources, and manage resources.
- The code which implements this part of tycoon is located under `src/Tycoon` path of Tycoon-gLite. These codes are implemented by Python script.
- **Auctioneers:** An auctioneer manages the resources on single providing host. Users use their agent to contact auctioneers to query the availability and current prices of resources, bid on resources, etc. To perform most operations on a providing host, a user must first set up a host account on that host. The auctioneer collects bids and manages the virtualization system to ensure that users receive the resources that they bid on.
- Auctioneer codes are in `src/Tycoon` and `src/Tycoon/Virtualization`. Auctioneer also is implemented by Python.
- **Service Location Service:** The SLS keeps track of which auctioneers are up and what their status is. There is an Internet-wide SLS available at `tycoon-sls.hpl.hp.com`.
- SLS also is implemented by Python and its code can be found under `src/Tycoon`.
- **Bank:** The bank keeps track of the amount of currency that different users have in their bank accounts. There is an Internet-wide bank available at `tycoon-bank.hpl.hp.com`.
- Bank is the only part which implemented by Java and placed under `src/bank`. It is recommended to use the Internet-wide bank, but for Tycoon-gLite the local bank should be installed.

Installing Tycoon can be done in two different ways: with the RPMs which can be downloaded from the hp lab (<http://tycoon.hpl.hp.com/~tycoon/dl/fedora/>) or from its source which can be cloned from the repositories.

```
# hg clone http://tycoon.hpl.hp.com/cgi-bin/hgwebdir.cgi/tycoon/KL
```

```
# hg clone http://tycoon.hpl.hp.com/cgi-bin/hgwebdir.cgi/tycoon/tycoon
```

To work with source code, two different packages should be cloned: tycoon and KL. tycoon consists of codes for the clients, auctioneer, bank, and service location service, and KL is library code for tycoon.



To test and work with source code, it is required to make a soft link from KL in Tycoon source path (suppose tycoon-glite is places in ~/tycoon-repo and KL is in ~/ tycoon-repo):

```
# cd ~/tycoon-repo/tycoon/src
# ln -s ../../KL KL
```

Tycoon has some dependencies which should be installed before working with Tycoon-gLite:

```
# yum -y install \
    mercurial \
    python-twisted.i386 \
    python-crypto \
    atlas \
    glpk-devel \
    pyOpenSSL \
    strace \
    sharutils

# wget -q http://tycoon.hpl.hp.com/~tycoon/dl/fedora/5/i386/cvxopt-
0.7.1-1.i386.rpm
# rpm -ivh cvxopt-0.7.1-1.i386.rpm
```

In above script, cvxopt is downloaded for Fedora core 5. Notice to use correct version for different distributions. Besides these, Java SDK also should be installed.

After installing the packages, some environment variables should be set. PYTHONPATH should set to the location of Tycoon-gLite source code and also JDKDIR should point to the location where Java SDK is installed.

```
# export PYTHONPATH=~/tycoon-repo/tycoon/src
# export JDKDIR=/usr/java/jdk1.5.0_08
```

The easiest way to copy Tycoon-gLite to file system is to create RPMs from source file and install them on file system. To do this, after going to the Tycoon-gLite execute the commands:

```
# make rpm
# make sub_rpms
# make kl_rpm
```

The first make creates tycoon-<version>.noarch.rpm. The second one creates tycoon_aucd-<version>.noarch.rpm, tycoon_aucd_plnm-<version>.noarch.rpm, tycoon_aucd_xen<version>.noarch.rpm and tycoon_client-<version>.noarch.rpm, and the last make creates KL-<version>.noarch.rpm.

2.3.3.2 Installing and Configuring Tycoon Client

The first step on working with Tycoon is installing client. Installing client can be done through its RPM from hp lab:

```
# yum -y -c http://tycoon.hpl.hp.com/~tycoon/dl/yum/tycoon.repo
install tycoon_client
```



This command by default installs Tycoon on Fedora Core 4. To install it on other version of Linux, we can use its source or find its prepared RPM files in the following address: <http://tycoon.hpl.hp.com/~tycoon/dl/fedora>.

The most important steps on working with Tycoon is generating public and private keys and creating an account on bank with generated public key. Public key is used for logging into hosts and performing bank operations. To generate it, use the ssh-keygen command:

```
# ssh-keygen -t dsa
# cat .ssh/id_dsa.pub >> .ssh/authorized_keys
# chmod 600 .ssh/authorized_keys
```

ssh-keygen -t dsa creates DSA public and private keys under .ssh/id_dsa.pub and .ssh/id_dsa files. The public key should be append to .ssh/authorized_keys for SSH connections.

The next step is to setup Tycoon configuration for user with its keys.

```
# tycoon user setup amir@kth.se amir .ssh/id_dsa.pub .ssh/id_dsa
```

This command creates a folder with name of amir@kth.se under ~/.tycoon and copies public and private keys on it with name of bank_account_public_key and bank_account_private_key.

The next step is creating an account on bank. This can be done in two different ways: creating account on public bank or do it on private one. These two are explained on section 4.3.4.

Tycoon uses different ports for different parts of it works. In general three important ports are available which should be opened on firewall for different reasons:

- SLS: Uses TCP port 25955 as outgoing
- Bank: Uses TCP port 8899 as outgoing
- Auctioneer: Uses TCP port 24571 as outgoing

2.3.3.3 Installing and Configuring Tycoon Auctioneer

An auctioneer manages the resources on single providing host. The auctioneer uses Xen as virtual machine to manages resources for different user. To do this the auctioneer should be installed on domain0 of Xen and then it can manage other domains for each user request.

To install auctioneer, again using yum can be helpful:

```
# yum -y -c http://tycoon.hpl.hp.com/~tycoon/dl/yum/tycoon.repo
install tycoon_aucd_xen3
```

This command installs auctioneer and also Xen kernel on Fedora Core 4. To install it on other version of Linux, we can use its source or find its prepared RPM files in the following address: <http://tycoon.hpl.hp.com/~tycoon/dl/fedora>.

To work with auctioneer, at first the system should be booted with Xen kernel. After installing auctioneer, it creates a script as /etc/rc.d/init.d/tycoon_aucd which starts auctioneer. Auctioneer listens on TCP port 24571 for requests.

One point which should consider while installing auctioneer with yum is that, it installs tycoon_file_system_xen3_fc4 file system image. This is the file system image which auctioneer uses for each request. For Tycoon-gLite the file system image should be created separately and replaced with this one (section 4.4).

Auctioneer requires some configuration to works well:



1. Changing the firewall setting:

```
# iptables -A INPUT -s \! 127.0.0.1 -p tcp --dport 8001:8002 -j REJECT
# iptables -A INPUT -s \! 127.0.0.1 -p tcp --dport 9601:9699 -j REJECT
# service iptables save
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

2. Copy the owner's bank key to /etc/tycoon with the following names: amir@kth.se_bank_account_public_key and amir@kth.se_bank_account_private_key. These keys are used by the auctioneer to make deposits into and withdraws from the owner's bank account.

3. Copy the owner's bank key to /etc/tycoon/admin_public_key. This key is used to authenticate remote administrative commands to the auctioneer.

4. Add or change the UserName option in /etc/tycoon/tycoon_aucd.conf to set the auctioneer's owner.

```
UserName = 'amir@kth.se'.
```

5. Now it's possible to start its service as follow:

```
/etc/init.d/tycoon_aucd start
```

After installing, configuring auctioneer and creating its file system image, the next step is to configure auctioneer to use the new file system image. As said before the auctioneer uses tycoon_file_system_xen3_fc4 as its file system image by default. This file system and related kernel is placed in /var/lib/tycoon/aucd/Xen3/lib, and the auctioneer uses a configuration file which its name is the name of user account in Tycoon and it is placed in /var/lib/tycoon/aucd/Xen3/accounts. For example if there is an account in Tycoon with name of amir, we can find amir.conf configuration file with the following format:

```
kernel = "/home/amir/vmlinuz-default"
disk   = ['file:/home/amir/default.ext3,sda1,w']
name   = "amir"
root   = "/dev/sda1 ro"
memory = 856
vif    = [ "mac=aa:00:00:77:66:d4, ip=10.106.212.209/30",
           "mac=aa:00:00:2b:ca:15" ]
ip     = "10.106.212.210"
netmask = "255.255.255.252"
gateway = "10.106.212.209"
hostname = "amir-boogiemann.pdc.kth.se"
```

To replace the default kernel and file system image it is enough to change the kernel and disk parameter in the configuration file and replace their value to new kernel and file system image path.

2.3.3.4 Installing and Configuring Tycoon Bank

Creating bank account can be done in two ways: using public bank or private one. Creating an account on public bank can be done by registering online from



<http://tycoon.hpl.hp.com/wiki/TycoonAccountForm> address. It requires public key and email address of the user to open an account with 100 credit. After opening an account, they inform with a reply message. To check the credit the following command can be used:

```
# tycoon bank get_balance
```

This command should return 100 as the base credit which was created for user.

It is recommended to work with the Internet wide bank at tycoon-bank.hpl.hp.com. But it is also possible to install a private bank. To install we should use the source of Tycoon. The following step then should be done to install private bank:

1. Set the PYTHONPATH and JDKDIR environment variables:

```
# export PYTHONPATH=~/.tycoon-repo/tycoon/src
# export JDKDIR=/usr/java/jdk1.5.0_08
```

2. Original Tycoon save banks data under `/var/lib/tycoon/tycoon_bank` but Tycoon-gLite use `/root/tycoon/tycoon_bank`. So this folder should be created:

```
mkdir -p /root/tycoon/tycoon_bank
```

3. Then build and start the bank:

```
TZ=UTC make -C ~/tycoon-setup/tycoon/src/bank
```

4. This command compiles the bank code and starts its service. It creates a file `accounts.data` which saves accounts on bank and also it creates two other files `privatekeys.dat` and `publickeys.dat` which consists of bank private and public keys. The bank uses a magic user `__Bank`, that can create accounts. But bank itself does not create `__Bank` folder under `~/tycoon`, so create `~/tycoon/__Bank` and copy `privatekeys.dat` and `publickeys.dat` under it and rename them to `bank_account_private_key` and `bank_account_public_key`. But notice to remove comment and also `__Bank=` form these files. Also copy the `bank_account_public_key` to `/etc/tycoon` with new name of `bank_public_key`.

5. The next step is to change the `/etc/tycoon/tycoon.conf` and change `BankURLList` and `BankPublicKeyFileName` from their default value to the following:

```
BankPublicKeyFileName = '/etc/tycoon/bank_public_key'
BankURLList = ('http://<your IP address>:8899',)
```

6. The configuration file `/etc/tycoon/tycoon.conf` is installed by installing tycoon RPM. Making Tycoon RPM as explained in 4.3.1 is created as follow:

```
# make rpm
# rpm -ivh tycoon-<version>.noarch.rpm
```

7. To create an account on installed bank use the following command:

```
# tycoon -oBankAccountName=__Bank bank create_account amir@kth.se
1000 ~/tycoon/amir@kth.se/bank_account_public_key
```

This command open an account on private bank with 1000 credit for Amir.



2.3.3.5 Installing and Configuring Tycoon SLS

As explained before, SLS keeps track of which auctioneers are up and what their status is. The easiest way to install Tycoon SLS is to use its RPM which can be found at:

http://tycoon.hpl.hp.com/~tycoon/dl/fedora/3/i386/tycoon_sls-0.5.0p53-1.noarch.rpm.

After installing this package a script can be found as `/etc/rc.d/init.d/tycoon_sls`. This script starts the SLS service with its default port 25955.

SLS was implemented by Python and its code can be found at `~/tycoon-repo/src/Tycoon/ServiceLocationService.py`.

There is a global SLS at `tycoon-sls.hpl.hp.com` which Tycoon by default is configured to use it. After installing the private SLS, the Tycoon configuration file (`/etc/tycoon/tycoon.conf`) should be modified to support it. To do this the following lines should be changed on `tycoon.conf`:

```
SLSHostName = "<SLS machine IP address>"  
SLSPortNumber = "25955"
```

In Tycoon-gLite it is required to install the private SLS for each site.

2.3.4 Creating gLite File system Image

A single file system image is used for both CEs and WNs. The image contains a Scientific Linux (SL) file system with the gLite CE and WN components pre-installed (but unconfigured). The procedure to install these components are the same as that of ordinary CEs and WNs with the exception that VO specific parameters are just assigned dummy values. These parameters are assigned their real values by the user (i.e. the VO manager) when the virtual cluster is created.

To do this at first install SL on QEMU or real partition, create a file system image (using `dd` command. More detail about how `dd` can be found in 4.2.2), and mount that image. In the remaining part of this section, it is considered that file system image is mounted under `/mnt`. The steps are as follow:

1. The first step after installing SL is installing gLite basic requirements.

At first, Java SDK 1.4.2_04 (or higher) should be installed. gLite finds it with variable `JAVA_LOCATION` in configuration file `site-info.def` (this file will be installed by `yaim`).

gLite uses `yaim` as a configuration tool. The latest version of it can be found at <http://grid-deployment.web.cern.ch/grid-deployment/gis/yaim>. The `yaim` configuration values are saved in a configuration file as key-value pairs. This file is shared among all the different node types (an example of configuration file is the file `/opt/glite/yaim/examples/site-info.def`).

2. `yaim` uses `apt` or `yum` to install other packages of gLite, (which can be selected from `site-info.def` file). It uses `apt` by default. To use `apt`, the address of repositories should be defined for it through `LCG_REPOSITORY` and `CA_REPOSITORY` in `configure` files.

```
LCG_REPOSITORY="rpm http://glitesoft.cern.ch/EGEE/gLite/APT/R3.0/  
rhel30 externals Release3.0 updates"
```

With using SL 3 add the following list in `/etc/apt/sources.list.d/cern.list` file:

```
rpm http://linuxsoft.cern.ch cern/slc30X/i386/apt os updates  
extras
```



```
rpm-src http://linuxsoft.cern.ch cern/slc30X/i386/apt os  
updates extras
```

But if using SL 4 the addresses should be change to:

```
rpm http://linuxsoft.cern.ch cern/slc4X/i386/apt os updates  
extras
```

```
rpm-src http://linuxsoft.cern.ch cern/slc4X/i386/apt os updates  
extras
```

And change the configuration of /etc/apt/preferences as follow

```
Package: *  
Pin: release o=linux.cern.ch  
Pin-Priority: 980
```

3. Yaim uses the install_node script to install the required nodes, which the format of using it, is like:

```
/opt/glite/yaim/scripts/install_node <site-configuration-file>  
<meta-package> [ <meta-package> ... ]
```

In Tycoon-gLite the file system consists of CE and WN without configuration. So to install them on file system image, the yaim install_node command can help:

```
# chroot /mnt /opt/glite/yaim/scripts/install_node site-info.def  
glite-CE TORQUE_server  
# chroot /mnt /opt/glite/yaim/scripts/install_node site-info.def  
WN_torque
```

Notice that only one of the above commands should be executed on one file system image. It means that one file system can be created to support CE or support WN.

4. Copy the root public key to file system image under /root/.ssh/authorized_keys to be able to do SSH to file system image.

```
# chroot /mnt mkdir -p -m 0700 /root/.ssh  
# echo `cat /root/.ssh/id_dsa.pub` >  
/mnt/root/.ssh/authorized_keys
```

5. The next step is creating voadmin user on image file system and copying some prepared keys and certificate to it.

```
# chroot /mnt mount /proc  
# chroot /mnt useradd voadmin  
# chroot /mnt umount /proc  
# chroot /mnt mkdir -p -m 0700 /home/voadmin/.ssh  
# chroot /mnt chown voadmin /home/voadmin/.ssh  
# echo "$SSHKEY" > /mnt/home/voadmin/.ssh/authorized_keys  
# chmod 644 /mnt/home/voadmin/.ssh/authorized_keys  
# chroot /mnt chown voadmin /home/voadmin/.ssh/authorized_keys
```

6. Because voadmin should have enough permission to configure system, the required permission should be added to it as sudo:



```
# echo "voadmin ALL=(root) NOPASSWD: /root/config_wn_set #  
voadmin" >> /mnt/etc/sudoers  
  
# echo "voadmin ALL=(root) NOPASSWD: /root/install_ce #  
voadmin" >> /mnt/etc/sudoers
```

7. Then the certificates should be copied to file system image. These certificates are used to configure CE or WN.

```
# mkdir -p /mnt/etc/grid-security/  
# cp $HOSTCERTDIR/*.pem /mnt/etc/grid-security/  
# chmod 400 /mnt/etc/grid-security/hostkey.pem
```

8. If the file system is CE file system the `install_ce` should be copied to file system and if it is WN file system, the `install_wn` should be copied. So depending the type of file system, one of the following command should be executed:

```
# cp -f ~/root/tycoon-  
repo/tycoon/packaging/glite/scripts/install_ce /mnt/root/  
  
# cp -f ~/root/tycoon-  
repo/tycoon/packaging/glite/scripts/install_wn /mnt/root/
```

Now file system image is ready to use.

2.4 FUTURE WORK

The following paragraphs explain weaknesses in the current implementation that should be addressed in future work.

The requirement that all nodes in a virtual cluster should reside on the same LAN is a great limitation. There are several ways to relax this requirement. One is to let the VO managers bid for additional “external” ports in the privileged domain, that are forwarded to the necessary ports of the VM when creating a WN, and then configure the CE to use the external ports and the IP address of the Auctioneer where the WN is installed.

Another solution is to set up a Virtual Private Network (VPN) between the CE and the WNs.

The fact that the VO managers have to manually specify host names, IP addresses and netmasks when creating new CEs, and host names when creating WNs gives rise to the following problems: 1) There is a risk that a new node receives the same name as an already existing node. Only a simple, DNS-based check is made to determine if a name already is in use or not. 2) The system gets unnecessarily complicated since VO managers need to receive information of unused IP addresses. 3) The IP address is actually a resource, but is not treated as such.

One solution to this is to let the VO managers bid on IP addresses and to let the Auctioneers decide host names.

On the other hand, if the LRMS could be configured to select a queue for a job depending on the VO the submitting user belongs to, it would be enough with one common CE to be used by all VOs. This is, however, to our knowledge not possible.



The requirement that there must be one SLS per LAN containing gLite-prepared Tycoon Auctioneers is a work-around for the problem of finding a set of Auctioneers that all reside on the same LAN. This is clearly not necessary if the “same-LAN-requirement” is relaxed, but it can also be removed adding a netmask field in the SLS information.

gLite uses DNS lookups (both reversed and direct) for host authentication. This makes it difficult to set up a NATed CE. We do not see the necessity of these lookups, and suggest that the possession of the private key corresponding to a host certificate should be sufficient as authentication.

The work-around for ensuring that no malicious code is executed when the configuration file is sourced, makes no guarantees for the code not to be executed at a later stage. A shell command like `${<var_name>}`, where `<var_name>` is a variable name, would execute the string contained in the value of the variable. We have not been able to find such code in the gLite system, but we think that alternative ways for specifying VO specific parameter values should be investigated.

2.5 TEST INSTALLATION

Fig. 3 presents the test installation of gLite-enabled Tycoon. It consists of the following elements:

- Bank and SLS services, located somewhere outside in the Internet,
- Clients and Auctioneer located on local PC, on privileged Xen domain. The host system for Xen was Fedora Core 4,
- Other Xen domains, created dynamically by Tycoon (pure FC4 as file system image without any gLite installation on it).

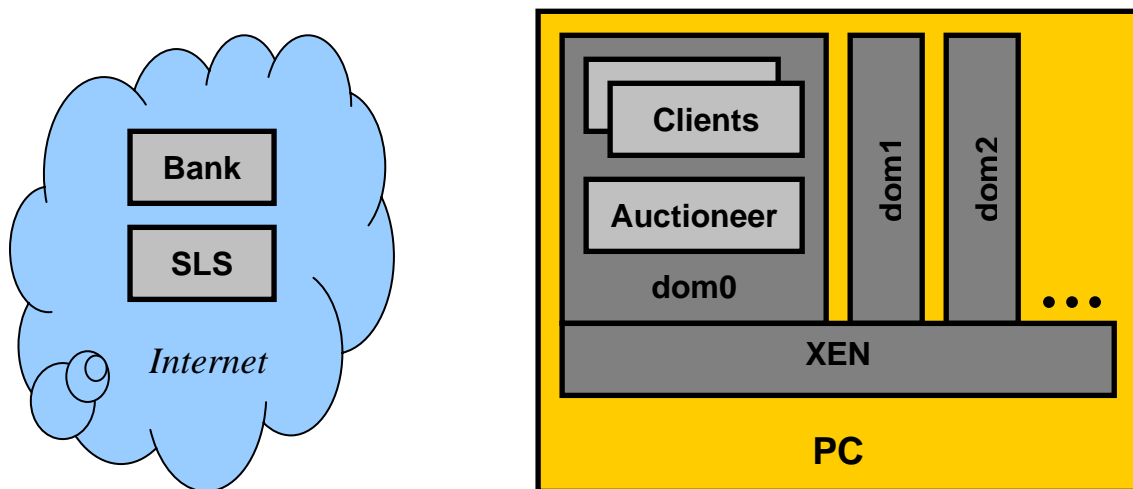


Fig. 3 Test Installation of Tycoon

3. DJRA1.3 - PROTOTYPE OF ACCOUNT MANAGEMENT SYSTEM

3.1 SYSTEM DESCRIPTION

The report [6] contains detailed requirements for user management in the BalticGrid. It describes also in detail existing software being subject for integration in order to fulfil these requirements – gLite Workload Management System and Virtual User System. Finally, it shows how these software may be integrated. These assumptions were base for the real implementation of the system.

The most important features of the system are:

- Virtual Organization based authorization (use VOMS service).
- Set of reusable local accounts (account pool) that may be allocated to the users for some time (session).
- Database, that stores history of sessions and any accounting and security information connected to the user.

The idea of such a system is presented on Fig. 4.

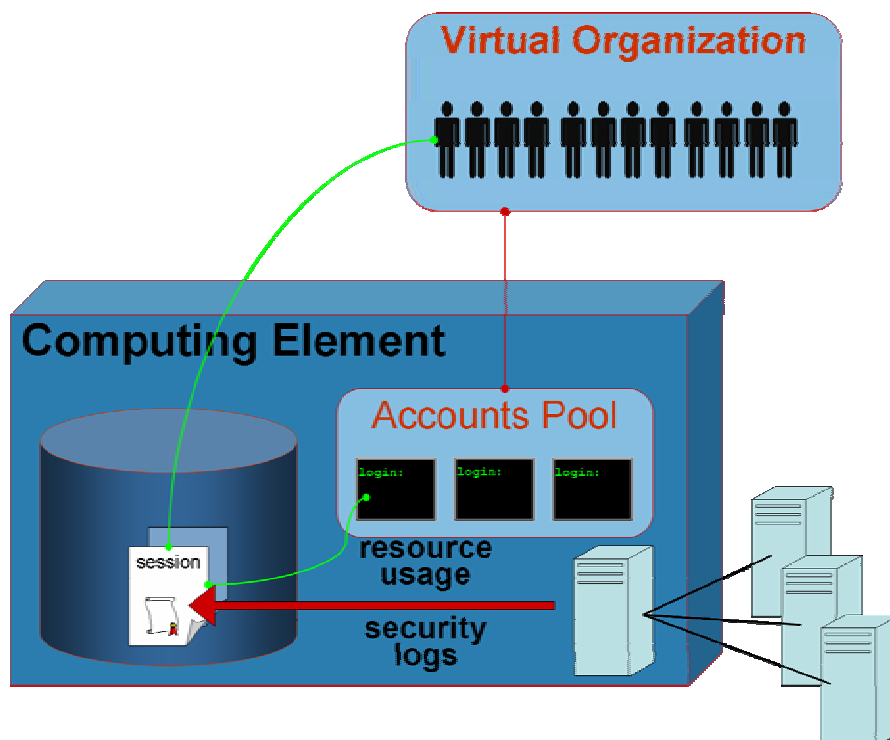


Fig. 4 Idea of User Management System

The user, while asking the resources on CE is first authorized. Typically, the authorization is based on Virtual Organization membership and VO attributes, although other methods are available. The user is mapped to an account from the accounts pool bound to the VO. This mapping –session is recorded in the database and the user’s job may start. During the job run, any accounting or security information may be stored in the database in the context of the session. The session may be automatically ended and user unmapped from the account if there is no activity on that account.



3.2 IMPLEMENTATION

3.2.1 Assumptions

The WMS System is responsible for the distribution and management of tasks across Grid resources. In the WMS user management on local resources is realized by modules called Local Center Authorization Service and Local Credential MAPPING Service. The main domain of VUS is also user management and the functionality of VUS and LCMAPS partially overlap. The functionality of VUS is wider and better fits the requirements and expectations of the BalticGrid administrators and users. However, VUS had to be enhanced to support VOMS attributes (groups, roles and capabilities) and the integration to the gLite had to be possibly seamless. The seamless integration, from the point of view of site administrator means the same pattern of configuration. Thus we decided to implement set of LCMAPS plugins that will interface the VUS functionality. The plugins follow the original LCMAPS pattern of authorization, including VOMS support, so the authorization configuration remains the same, but they offer an additional VUS features.

3.2.2 Architecture

Fig. 5 demonstrates how VUS was integrated with the Workload Management System on the Computing Element. An important part of the CE software in the gLite architecture is the Globus Gatekeeper. The authorization and mapping a Grid identity to a local identity (account) tasks of the Gatekeeper are performed by LCAS/LCMAPS. LCAS performs an authorization task by querying set of plugins. Then LCMAPS performs the mapping task by querying set of plugins. Set of VUS-enabled LCMAPS plugins will be implemented:

- local account (user will be mapped to a local account)
- pool account (user will be mapped to one of the pool accounts – VUS virtual accounts)
- VOMS local account (user will be mapped to a local account, authorization is based on VOMS attributes)
- VOMS pool account (user will be mapped to one of the pool accounts, authorization is based on VOMS attributes – VUS virtual accounts)

VUS replaces LCMAPS gridmap-dir functionality and manages the mappings via its database. The database stores both current and historical mappings (called sessions) and allows for storing any accounting or logging data in the context of the session. The session may be automatically finished by VUS, if the mapped account is no longer used. Then, the virtual account is released and may be mapped to another user (new session is opened).

The Accounting Module cooperates with DGAS Gianduia in order to process the accounting data gathered in the database. It will be implemented as a Gianduia sensor. The database is supplied with accounting and auditing data by any number of external programs (Loggers), that may be run on request or cyclically.

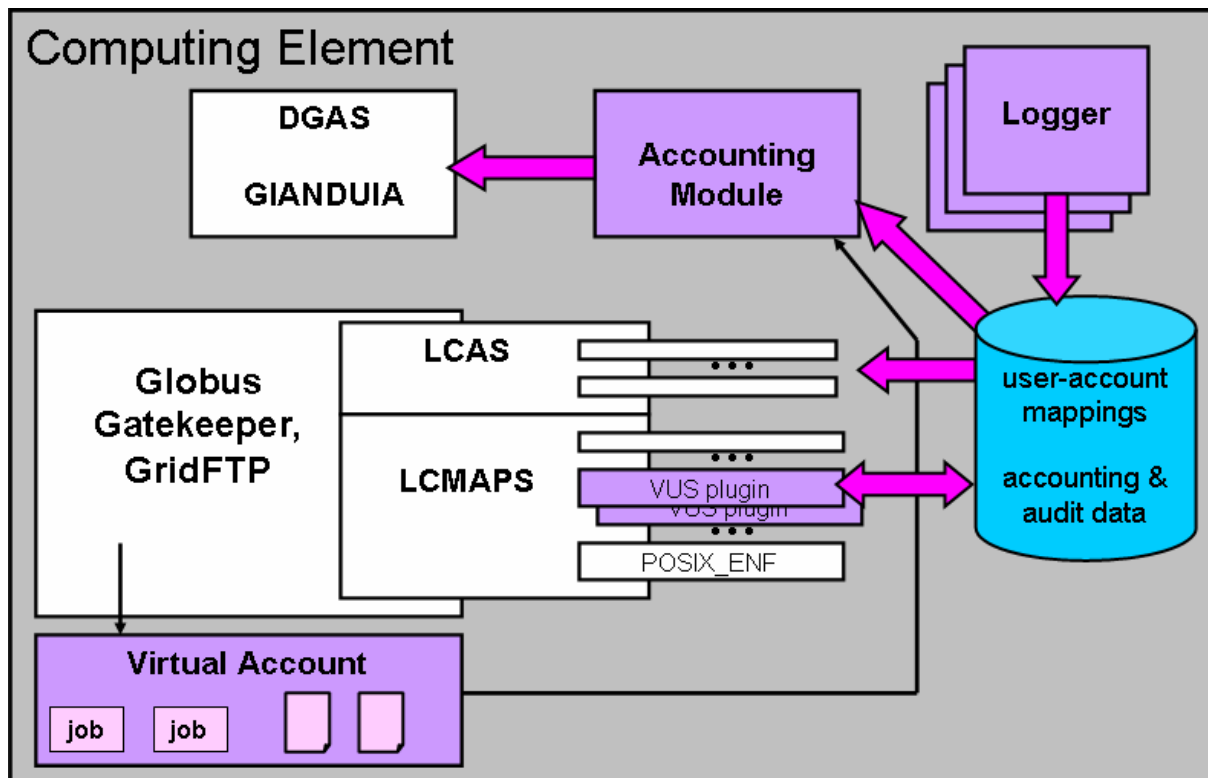


Fig. 5. Architecture of the User Management System

3.2.3 VOMS Attributes

The VOMS proxy certificate contains uses X.509 Attribute Certificates (AC) to bind a set of authorization attributes like VOMS group membership, user role and capabilities. The AC is signed by the VOMS service and located in the non-critical section of the proxy, so the certificate may be successfully processed even by “VOMS-unaware” CEs. The VOMS attributes are stored in the Fully Qualified Attribute Name (see [12]):

```
<group name>/Role=[<role name>][/Capability=<capability name>]
```

3.2.4 Authorization.

The LCMAPS reads the local site policy from the file (lcmaps.db) that describes the authorization policies (order of loading of plugins and their parameters). The policies are evaluated until a policy evaluation returns a true result. Most of the plugins (specifically all VUS plugins described above) use generalized gridmap files. Such files define mapping user DN or VOMS VO-FQAN (possibly using wildcards) to a local accounts or accounts from a pool. Then, a special plugin enforces the authorization decision by setting user Unix UID and GIDs.

3.2.5 VUS Database

Fig. 6 contains a simplified VUS database schema. The database consists of the following tables:

- **User** – grid users that have ever requested the local resources, identified by their distinguished names (subject of the certificate).
- **Account** – local operating system accounts that have been used for the grid users. The relation between User and Account represents the current mapping.



- **VO** – Virtual Organizations of the users.
- **Session** – the table contains the history of the user – account mappings. The mapping is stored with context: VO of the user, FQAN, accounting and audit data.
- **Resource_type** – accounting metrics, e.g. “CPU time”. May be used for defining any resource types both standard and non-standard.
- **Accounting_data** – the real amount of resources used.
- **Event** – any text messages that may be subject for audit. Here these messages, usually stored in system logs in the context of OS account (local user identity), have an additional context of user DN, VO, FQAN (global user identity).

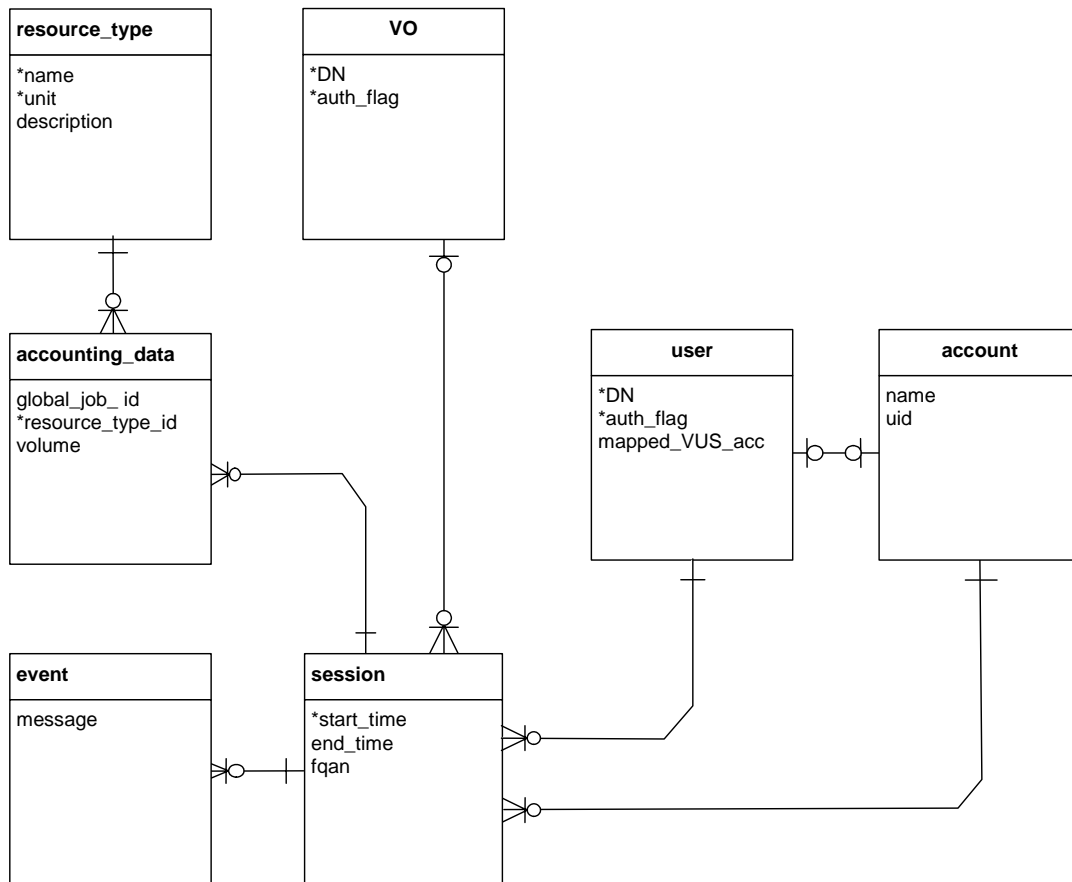


Fig. 6. VUS Database

3.2.6 Use Scenario

1. The user calls voms-proxy-init, specifying VOMS and possibly also group, role and capabilities he needs.
2. The VOMS service signs the VOMS extension and the user gets the proxy.
3. The user requests the job run via Workload Manager.
4. WM contacts the Gatekeeper service on the Computing Element and sends the request of the job.
5. The user is authorized by LCAS and LCMAPS with a set of VUS authorization plugins.



6. If the user is authorized by “pool-like” plugin, he will be mapped to one of virtual accounts that match the pattern fro gridmap file (e.g. VO and FQAN). GIDs are set accordingly.
7. The job is scheduled to the local scheduler and distributed to Worker Nodes.
8. The user application may write any non-standard accounting data to the VUS database.
9. The security warnings, if occur, may be written to the VUS database.
10. When the job is finished, the Gianduaia collects accounting data from its sensors, including the Accounting Module of VUS.
11. The user gets the results of the job.
12. DGAS collects the accounting on HLR servers and may perform usage cost computation.

3.3 INSTALLATION AND CONFIGURATION

3.3.1 Installation and Initial Configuration

Prerequisites

1. Installation of Computing Element.
2. MySQL (other SQL DBMS possible, but not tested).
3. STDC++ library (libstdc++ RPM).
4. UNIX ODBC with driver for MySQL (may be installed and configured by VUS configuration script if not done before).

Download

From the location <http://www.balticgrid.org/Internal/JRA1/vus4glite1> one should download the following:

1. **glite-security-lcmaps-plugins-vus-basic-0.9.1-1.i386.rpm** – the VUS enabled LCMAPS plugins.
2. **glite-security-vus-common-0.9.1-0.i386.rpm** – the base library for VUS.
3. **vus-config.tar.gz** – configuration scripts and instructions

Installation

As root, install the VUS RPMs:

```
rpm -i glite-security-vus-common-0.9.1-0.i386.rpm
```

```
rpm -i glite-security-lcmaps-plugins-vus-basic-0.9.1-1.i386.rpm
```

3.3.2 Initial Configuration

Unpack the archive `vus-config.tar.gz` and change to the unpacked directory.

Become root and check if `$GLITE_LOCATION` is set correctly.

Run `vus-config.sh`. The interactive script will ask you set of questions, the default values (displayed in brackets) you may set by pressing Enter. The script performs the following actions (each action may be skipped if already performed, the script may be stopped at any moment by Ctrl+C):

1. Creating VUS database user and database.
2. Creating empty tables in the database.
3. Installing ODBC database access tools (UnixODBC and MySQL driver).



4. Configuring ODBC database access (definition of ODBC data source).
5. Configuring VUS -the configuration file `$GLITE_LOCATION/etc/vus.conf` is created, the old configuration is removed.
6. Configuring LCMAPS (`$GLITE_LOCATION/etc/lcmaps`) -this affects how the authorisation is performed. Your current configuration will be completely overwritten, so you may have to perform this step manually (see next section). The old configuration is stored respectively in `lcmaps.db.2` and `lcmaps.db.gridftp.2`.

3.3.3 Advanced Configuration

ODBC

See <http://www.unixodbc.org> for more details.

VUS

The configuration file is `$GLITE_LOCATION/etc/vus.conf`. Configuration parameters are presented in Tab. 1.

VUSDataBase	VUS database name (ODBC data source name)
VUSDataBaseUser	database user name
VUSDataBaseAuth	password to the database
VUSMapTimeout	the minimal user mapping time in seconds
ClearAccountScript	path to the script, that is run on the account while unmapping user, may be used for removing any personal data left by the user.

Tab. 1 VUS Configuration Parameters

IMPORTANT: it is strictly recommended to set ownership of `vus.conf` to root and access rights to 600 for the security reasons.

LCMAPS

File `$GLITE_LOCATION/etc/lcmaps/lcmaps.db` defines authorisation policies for Gatekeeper.

File `$GLITE_LOCATION/ETC/LCMAPS/LCMAPS.DB.GRIDFTP` defines authorisation policies for GridFTP.

The standard LCMAPS plugins in the above files may be replaced by corresponding VUS-enabled ones Tab. 2 presents the plugins.

Standard LCMAPS plugin	Corresponding VUS enabled plugin	User mapped to ...
<code>lcmaps_localaccount.mod</code>	<code>lcmaps_vus_localaccount.mod</code>	local -personal account, exactly as in <code>gridmap-file</code>
<code>lcmaps_poolaccount.mod</code>	<code>lcmaps_vus_poolaccount.mod</code>	one of the pool accounts – VUS virtual accounts
<code>lcmaps_voms_localaccount.mod</code>	<code>lcmaps_vus_voms_localaccount.mod</code>	local account, authorization is based on VOMS attributes, users of matching VOMS attributes are mapped to the same account
<code>lcmaps_voms_poolaccount.mod</code>	<code>lcmaps_vus_voms_poolaccount.mod</code>	one of the pool accounts, authorization is based on VOMS

	attributes – VUS virtual accounts
--	-----------------------------------

Tab. 2 LCMAPS plugins

The VUS-enabled plugins use the same arguments as their standard counterparts, except `-override_inconsistency`.

The simplest way to enable VUS on your site without modifying the authorisation policy is to do the above replacement and removing the unsupported argument.

More details on: <http://www.nikhef.nl/grid/lcaslcmaps>

3.4 FUTURE WORK

The following points are planned to be realised in the next release:

1. Integration with DGAS accounting component of gLite.
2. Enhancements to the configuration tools, to use the same pattern as LCG/gLite.

3.5 TEST INSTALLATION

Fig. 7 presents the test installation of integrated VUS and gLite. It consists of the following elements:

- VOMS service from the BalticGrid,
- User Interface (UI) located on local PC (Pentium IV machine with Fedora Core 4),
- Workload Manager (WM) + Logging and Bookkeeping (LB) services located on VMWare Virtual Machine with Scientific Linux 3 on the same PC.
- CE with VUS software + Worker Node (WN) located on VMWare VM with Scientific Linux 3 on the same PC.

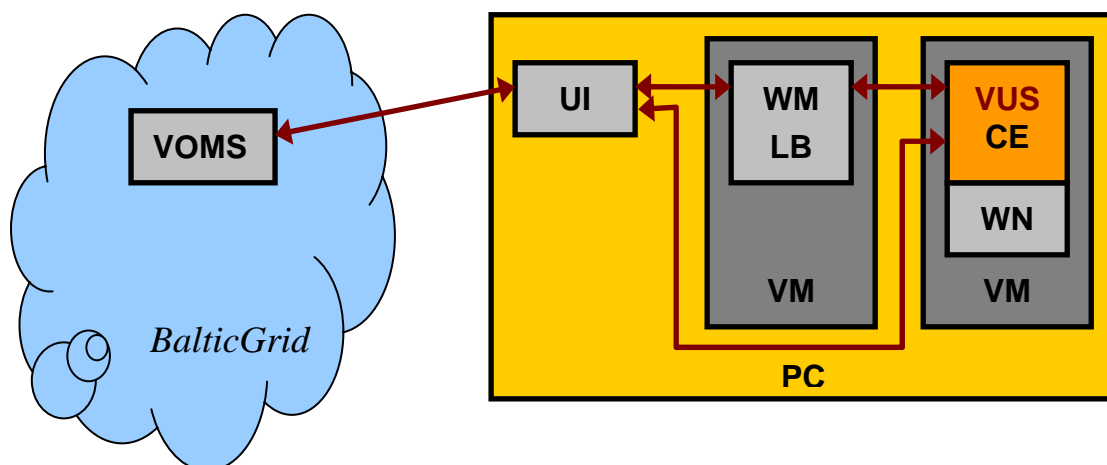


Fig. 7 Test Installation of Virtual User System

Additionally different certificates were used for tests:

- official user certificate, registered on BalticGrid VOMS –to test VOMS attributes,
- few certificates signed by test Simple CA – to test parallel requests.



4. CONCLUSIONS

Tycoon market-based Utility Computing infrastructure was used to implement and manage a SLA-capable execution environment hosting unmodified gLite Computing Elements. Virtual User System was integrated with gLite in order to fulfil BalticGrid requirements on the area of the user management. The both systems were discussed in detail in this document. The basics of the systems were described and their architectures overviewed. A special attention was put on installation and configuration procedures, as the document aims to help administrators to work with these systems.

The both systems were installed and tested in a very limited environments. Now, they are matured enough to start deployment in a wider testbed. So that, the next step will be installation of the software on sites managed by all JRA1 partners.

The document lists limitations and possible improvements to the current version of the software. These will be elaborated in parallel to testbed deployment phase.



REFERENCES

- [1] Kevin Lai, Lars Rasmusson, Eytan Adar, Stephen Sorkin, Li Zhang and Bernardo A. Huberman, *Tycoon: an Implementation of a Distributed Market-Based Resource Allocation System*, Proceedings of the ACM Conference on Electronic Commerce (2004)
- [2] Kevin Lai, *Markets are Dead, Long Live Markets*, HP Labs, Palo Alto, CA, USA (2005)
- [3] Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., and Neugebauer, R. *Xen and the Art of Virtualization*. In Proceedings of the ACM Symposium on Operating Systems Principles (2003).
- [4] *gLite Installation and Configuration Guide v. 3.0 (rev.2)*, http://glite.web.cern.ch/glite/packages/R3.0/R20060502/doc/installation_guide_3.0-2.pdf, May 8 (2006)
- [5] *BalticGrid, EU FP6, Contract 026715, Annex I - "Description of Work"*, Sep. 15 (2005)
- [6] *Account Management Report on Integration with Workload Management System (WMS) Environment*, BalticGrid deliverable DJRA1.1., June 2006.
- [7] <http://www.globus.org/toolkit/>
- [8] *EGEE User's Guide, WMS Service*, <https://edms.cern.ch/document/572489/>, May 3, 2006.
- [9] <http://www.unixodbc.org>
- [10] <http://www.nikhef.nl/grid/lcaslcmaps/>
- [11] R.Alfieri, R.Cecchini, V.Ciaschini, L.Dell'Agnello, A.Frohner, A.Gianoli, K.Lentey, F.Spataro, *VOMS: an Authorization System for Virtual Organizations*, 1st European Across Grids Conference, Santiago de Compostela, February 13-14, 2003.
- [12] V.Ciaschini, *A VOMS Attribute Certificate Profile for Authorization*, <http://infnforge.cnaf.infn.it>, October 2004.
- [13] M.Jankowski, P.Wolniewicz, N.Meyer, *Virtual User System for Globus based grids*, Cracow Grid Workshop '04 Proceedings, Cracow 2004.