# Installing and Configuring Application Software on the LHC Computing Grid

Roberto Santinelli, Flavia Donno
*CERN, Geneva, Switzerland*
*Roberto.Santinelli@cern.ch*, *Flavia.Donno@cern.ch*

## Abstract

*The management of application software is major scientific and practical challenge for designers of large-scale production Grids. The LHC Computing Grid is unique in the sense that coupling between application scientists and the resource providers is extremely loose, thus adding even more complexity to the software management problem. After an analysis of the requirements for a Grid software management service from users' and site administrators' perspective, we give an overview of the solution adopted by the LHC Grid infrastructure to support High Energy Physics experiments, highlighting features and current limitations. Tank&Spark is our proposed solution based on P2P technology that extends the LHC Grid application software system and tackles some of its limitations. Tank&Spark can be used as a stand-alone service also in other Grid infrastructures. Here we illustrate the design, deployment and preliminary results obtained.*

## 1. Introduction

The problem of application software installation and configuration in Grid-aware computing facilities is not trivial. This is due to several reasons: large number of sites at which the software has to be promptly maintained; different ownership and hence configurations and policies at the sites; hardware and software heterogeneity of the systems. The Grid middleware is generally installed by system administrators at a site level via customized tools which can be coupled with or decoupled from the central system management facility. This allows for basic self-contained applications, such as an executable that runs under a specific flavor/version of Linux, to be executed on a Grid.

In a more complex situation, a scientist wants to execute her domain specific application code on the Grid. In such a case, if the application has a modest size and is self-contained (no external library dependencies), it can be shipped with the request for execution to a computing resource and then removed once it has terminated. However, in most cases, dependencies on external packages and specific system configuration (such as support for MPI, compiler versions, etc.) might generate problems. One can think that the execution environment needed to run an application can be specified within the job description so that the Grid can find the appropriate resources. However, the schema describing either the resources or the application requirements might not be sufficient as to allow the Grid system to correctly match requirements and resources availability, therefore ensuring a correct execution environment to the application.

If the applications needed by the user are of considerable size (tens of Gigabytes), it is quite inefficient and sometimes even not possible (not enough storage available on the computing node) to ship the application together with the user request.

To overcome the issues described above a Virtual Organization (VO) can then provide special resources to run VO specific applications. On such resources an appropriate execution environment is guaranteed to VO users, either by ensuring an appropriate system environment or by making available pre-installed VO specific software applications.

An important requirement in order to allow for a correct execution environment is to guarantee that the software provided is well configured and validated, i.e. it runs smoothly and produces the expected results.

It is quite reasonable for a site to satisfy specific network or system configuration requirements imposed by a VO. However, if a site supports more than one VO, it is impractical if not impossible for an administrator to install, configure and validate VO specific software while keeping up with software releases and installation requirements. Often many releases of the same package (libraries, object and source files, and other auxiliary files) need to coexist. Site administrators might not be familiar with all VO software requirements and release procedures. For this reason “privileged” users within a VO are sometimes identified and made responsible for managing application software in a Grid. To achieve this, such a VO software manager requires adequate tools that allow for triggering software installation on Grid, managing VO disk space and software versions, planning for software upgrade and removal, publishing site and software status related information, etc. From a site administrator’s point of view what is described above becomes a source of concern in terms of site security, local policies to be respected, maintenance scheduling and related problems, etc.

Therefore, many issues need answers and solutions, such as:

- Establishing a mechanism that allows for scheduling a software installation process when appropriate.
- Ensuring adequate disk and space management.
- Failure and conflict resilience.
- Resolving software dependency issues.
- Handling concurrent installation processes.
- Satisfying pre-requisites before the application software installation is triggered.

The above is a non-exhaustive list of issues that has to be considered when designing a service for managing application software installation and configuration on the Grid.

In the LHC Computing Grid (LCG) [1] infrastructure created to support High Energy Physics experiments, we have been trying to tackle this problem for quite some time.

In this paper we provide an analysis of the requirements for a Grid application software management service from users’ and site administrators’ perspectives. We give an overview of the solution adopted by the LHC Grid infrastructure to support High Energy Physics experiments, highlighting features and current limitations. Then we present Tank&Spark, our proposed solution based on P2P technology that extends the LHC Grid application software system and tackles some of its limitations. Tank&Spark allows for centrally triggering and controlling software installations at many remote sites. It is very flexible and has been designed to use plug-ins for authentication, authorization, storage management and back-end databases. Therefore, it is a generic service that can be used on a generic Grid infrastructure with few adaptations.

This paper is structured as follows. In Section 2 we give an overview of the LCG infrastructure, describe the solution adopted for application software management and the feedback received. In Section 3 we list the requirements collected from users and site administrators in the attempt to design an adequate solution for LCG. In Section 4 we illustrate our proposed solution currently under test and give some implementation details. Preliminary results using the new software installation service are given in section 5. A summary on related and future works and conclusions are presented at the end of this article.

## 2. The LHC Computing Grid

The Large Hadron Collider (LHC) is being currently being built at CERN near Geneva. Four main experiments (ALICE, ATLAS, CMS and LHCb) with a total of some 6,000 physicists coming from several hundred universities or laboratories from around the world will be involved in the analysis of the data that will be generated by the collision of accelerated particles to understand the early times of the universe. When it begins operations in 2007, the LHC machine will produce roughly 15 Petabytes of data annually and will be operational for about 15 years. The data from the LHC experiments will be distributed according to a four –tiered model. A primary backup will be recorded on tape at CERN, the "Tier-0" centre of LCG. After initial processing, this data will be distributed to a series of Tier-1 centers, large computer centers with sufficient storage capacity for a large fraction of the data, and with round-the-clock support for the Grid.

The mission of the LCG project is to build and maintain a storage and analysis infrastructure for the entire high-energy physics community that will use this machine.

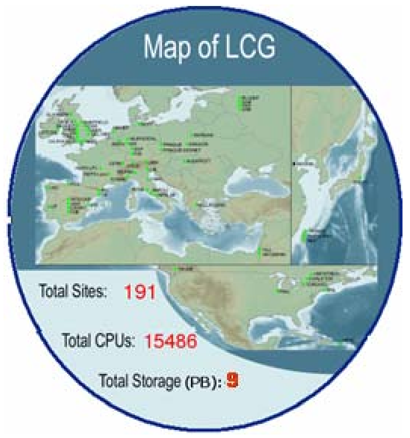

**Figure 1:** The LCG Infrastructure

The infrastructure consists of a large number of geographically distributed resources and services made accessible to users organized in Virtual Organizations.

It currently consists of 191 sites, with 15,486 CPUs and 9 PB of storage. Figure 1 gives an overview of the geographical distribution of LCG resources.

## 2.1. The LCG Middleware

The LCG middleware is based on software supplied by several other projects to manage authentication and authorization, job submission and management and data distribution and access. These projects include Globus [2], Condor [3], the Virtual Data Toolkit [4], the European DataGrid project[5], the gLite toolkit [6] and other services developed by LCG experts.

**2.1.1. Authentication and Authorization**. Authentication services are provided by Globus and based on the X.509 Grid Security Infrastructure (GSI) [7]. User credentials of limited lifetime (proxies) can be automatically renewed through a Proxy Service (PS).

LCG has developed special bridge services to allow users with a valid proxy to obtain a Kerberos token as well to access special AFS areas.

The Virtual Organization Management Service (VOMS) allows for the definition of roles in the extended user proxy. If a Grid service is interfaced to the VOMS, then a particular user role gets honored or the user is mapped to a privileged special account enabled locally for specific operations.

**2.1.2. Job Management.** The Job Management Service, better known as Workload Management System (WMS), is responsible for the management and monitoring of user jobs submitted from a special Grid gateway client known as User Interface (UI). Resource status and characteristics are published in the LCG Information System (BDII). The Resource Broker (RB) machine is responsible for: matching job requirements to resources; scheduling jobs for execution through Condor-G to an appropriate batch queue known as Computing Element (CE); tracking the status of jobs; retrieving the job output. The Worker Nodes (WNs) are computing machines behind the CE in charge of executing the job.

**2.1.2. Data Management.** The Data Management System (DMS) provides services for data movement and replication and for cataloging of file locations. Client applications and APIs available on the UI and the WNs allow user jobs to manage data in the Grid. Data is stored on Storage Elements (SE), which can provide disk or tape based storage back-ends. The GridFTP protocol is the most commonly used for data transfer.

**2.1.3. Information System.** The Information System or BDII stores information about resource status and characteristics, middleware and application software available at a site and their correspondent releases. This is the place where a site publishes information about the VOs supported locally and the local configuration of system services (outbound connectivity, MPI support, resources hardware characteristics and their power, etc.). Information is published following a specific schema that goes under the name of GLUE [8].

**2.1.4. The VO Box.** Recently, a new service has been introduced in LCG, the VO Box. This Service allows VOs to install, run and control specific long-lived VO agents which execute in a secure and controlled environment. The VO Box allows a direct login via gsissh for special users of the corresponding VO and the registration of a proxy for an automatic renewal. The VO Box has been provided to allow for VO specific services that the LCG middleware does not yet provide, such as a Data Subscription Service that allows a site to subscribe for receiving a subset of the data produced at another site, as soon as it becomes available. We will describe how this service is being used to improve the Application Software Installation and Management Service in LCG.

## 2.2. LCG Application Software Installation and Management Service

In LCG Gigabytes of VO specific software or frequently changing user applications need to be pre-installed, configured and validated before a user job is executed at a site.

In Table 1 we report the space requirements and frequency of updates of application software for the LHC experiments.

| **Exp.** | **Space requirements** | **Frequency of updates** | **Concurrent releases** |
|---|---|---|---|
| Alice | 1-2 GB | 3/month | 3 |
| ATLAS | 6 GB | 4-6/month | 3 |
| CMS | 2 GB | 6/month | 3 |
| LHCb | 1-2 GB | 4/month | 2 |

**Table 1:** Space requirements, frequency of updates and number of concurrent supported releases per LHC experiment.

Following the requirements imposed by the experiments, in LCG Experiment Software Managers (ESM) are designated people with privileges for completely managing software for a specific VO on a per site basis. Each experiment selects one or more ESMs. The ESM is the person in charge to install, configure, validate and update the software of the experiment at each site.

An ESM can also publish univocally identified software tags in the Information System to announce the availability of a specific software version at a site. Via the published tag users can then select sites to run their jobs.

The X.509 certificate subject of an ESM is mapped to a local account with special write privileges in designated experiment areas.

The experiment software is first packaged into a software specific bundle and moved to one of the SEs belonging to the site where the software will be installed via the Grid DMS. The software can be installed on the local farm using a local cache on the SE. Since the SE and the WNs are on the same local area network no inbound/outbound network connectivity requirement from the WNs is imposed by the system. The bundle contains scripts to validate the installation, i.e. to verify that the installation has been executed with success and the software produces the expected results.

The ESM directs an *installation job* to the site using the WMS from a UI. Once arrived on the WN, the job installs the software at a location specified by the value of the variable VO_<EXP>_SW_DIR. The content of this variable is essential to determine the behaviour of the installation service:

- If the value is ".", the software is installed and validated in the working area of the job and then removed when the job is finished. If the validation step has passed with success, the ESM can publish the attribute *GlueHostApplicationSoftwareRunTimeEnvironment* in the Information System. Such an attribute is set to a VO software specific tag that certifies the site for that specific version of the software. Subsequent jobs ending up on the same WN for execution have to perform first the software installation step in their working area and then execute the real job.
- If the value of the VO_<EXP>_SW_DIR environmental variable is not ".", the software is installed in a permanent area that is shared among the WNs. The ESM job has to first check if the version of the software to be installed is already present. Only if that version is not there, the job proceeds with the installation since the ESM is guaranteed to have write privileges in that area. In this case the ESM can run validation scripts in a second step, and only if the validation process is successful, the ESM can publish the relative software tag in the BDII using provided tools.

The solution adopted by LCG is largely serving its purpose. In particular, it provides a framework within which experiments are free to use their own proprietary distribution tools (Pacman [9], tarballs, DAR [10], or even a CVS repository accessible via http, if outbound connectivity from the WNs is available). However, the current solution has several limitations, as reported by the LHC experiments in [11]:

- The lack of "roles" severely constrains the abilities of software managers. An ESM should be able to dynamically switch his/her role and become a normal user able to submit normal user requests to the Grid.
- Many jobs failures are often due to loss of visibility of the NFS file system either during software installation or during run. Avoiding the use of NFS on a large installation can cure this problem. However, with the current system it is impossible to trigger an installation on a whole farm of WNs on demand.
- The ESM job has to compete with normal user jobs without any special priority.
- There is no automatic mechanism to trigger a software installation on the whole Grid, i.e. on all sites supporting a specific VO.

## 3. The requirements for a Software Management Service

In order to test the complete functionality of the LCG infrastructure before the start of the LHC accelerator operations and to understand reliability and scalability issues, a series of Data Challenges (DCs) were performed on LCG. During such DCs simulated data files were produced and injected to the Grid and the entire data flow and analysis infrastructure has been tested. This was a great opportunity for us to understand the problems and collect requirements in terms of software installation from the four LHC experiments and from the site administrators running the LCG facility. Here, we list the main features that a software installation service should provide from a user and site administrator perspective.

### 3.1. The User's and Experiment's View

Interacting with users and ESMs and supporting them during the data challenges, we highlighted the following needs:

- Each LHC experiment requires frequent updates of software releases, about three times per month. Software should be installed without special interaction with site administrators. Whenever necessary, old unused versions should be removed.
- Several releases of the software should coexist at a site.
- All software for the experiment should be relocatable. The root installation path should be accessible through an environment variable.
- No root access should be required to install experiment software.
- Only a subset of users should be entitled to manage experiment software. This is achieved in LCG through the ESM role. An ESM should be able to add/remove software at any time without communication with the site managers.
- The software has to be accessible through standard POSIX I/O calls on the WN.
- The ESM should be able to install software on a per site base as well as launching a request to the entire Grid supporting the specific VO.
- The ESM should be able to verify the installation in separate steps. Different kinds of validation procedures can be run by the ESM at different moments.
- The ESM should have the possibility to publish special software tags in the Grid Information System. This is done in order to advertise all installed and validated versions of the software available at a site allowing users to direct jobs to that site.
- It is the responsibility of the ESM to prepare a given software distribution for a given release. Dependencies should be completely managed and fully satisfied.
- Experiment software can be packaged as the experiment requires: tarballs, RPMs, DAR files, Pacman distributions, etc. Installation and validation scripts should be provided by the experiments. Therefore, dependencies should be expressed in a way that those scripts can process them.
- The user environment should be setup by a script placed in a given location that the user job sources as a very first step.

A more detailed report about user's and experiment's requirement has been produced by the LCG GAG team in [12].

### 3.2. The Site Administrator's View

The main concerns from the site manager's point of view are summarized in the following list. Such requirements came out after a survey conducted among the sites participating to LCG [13],[14].

- For security and maintenance reasons, no daemons running on WNs should be allowed. Neither user applications nor Grid software installation services should have control over WNs.
- Every individual access to a site should be traceable. For this reason, no shared accounts are allowed.
- The information published by a site should not be corruptible.
- Service actions, such as restart, flushing, etc. should not be triggerable externally unless policies can be applied. In fact, this could lead to denial-of-service (DoS) attacks when the service is continuously restarted. Such a DoS attack not only affects the VO with the compromised ESM account, but will also bring the entire site down.
- Access to any tool/service should be strongly authenticated, and the restrictions and policies should be applied on the server-end and not on the client-end.
- Possible inbound/outbound connectivity requirements from WNs should be avoided.
- It should be possible to apply and enforce site policies to the software installation mechanism.
- One should not assume shared file systems among WNs to serve experiment software. Such a requirement in fact poses serious performance, reliability and scalability problems for large installations.

## 4. Our Solution: Tank&Spark

With one of the recent releases of the LCG middleware we introduced a new service that site administrators can optionally configure and activate. Such a service called **Tank&Spark** satisfies the requirements previously listed. The toolkit is fully integrated with the software management system in use in LCG. However, it has been designed to work as well as a standalone service in a generic Grid infrastructure, requiring only a few modifications. The toolkit provides as well for many other interesting features, as explained later, and it enforces site policies defined by site administrators.

Triggering automatic software distribution to the Grid can be achieved via a Grid job or directly via contacting the installation service at a site from a User Interface (UI). In this latter case the ESM will not compete with normal user jobs but it can immediately schedule a software installation request.

The architecture of such a service foresees a multi-threaded server *(Tank)* running on a Grid machine (such as a CE), a client application *(Spark)* that runs as cron job on each WN of a farm or in a VO Box, and a r-sync server running on a disk-server (a Storage Element) acting as central repository of the experiment software.

Tank is a daemon listening on a dedicated port for incoming connections. It can currently accept GSI-authenticated and insecure connections but other security protocols can be easily integrated. The service is also integrated with the VOMS and uses user credentials interpreting user roles. Tank uses a MySQL database to store internal status information.

The server component represents the central intelligence of the system managing the various releases of the experiment software that need to be installed/removed.

The server enforces local policies set by the site administrator: he/she can describe whether and when the installation/removal process can take place. If Spark is invoked on a WN via a user job, it transfers the VO software bundle to the local SE, it installs and validates the software (if not already installed) starting from that bundle in the area pointed to by the environment variable VO_<EXP>_SW_DIR, triggers an r-sync installation of that software to the central repository on the SE and then contacts Tank. The r-sync software synchronization is currently not secure. The r-sync server can only be contacted locally. After authenticating with Tank, Spark registers the new software tag in Tank's DB.

From a UI a user can also invoke Spark installed on a VO Box. In this case the user bypasses the WMS and can immediately schedule software installation. In this case, user credentials are passed via an ssh tunnel. As done in the previous case, Spark triggers the copy of the software bundle on the SE, installs the software locally, synchronizes it in the local repository and then it contacts Tank, presenting the user credentials. Then the process proceeds as in the previous case.

On the other WNs, the client program (Spark) is invoked by a cron job running every 5 minutes. It retrieves the list of tags relative to software releases installed since the last update on that machine. In case of new updates, the client synchronizes the local software area with the central repository. The location of the central repository per VO is defined by the Tank and Spark configuration file.

Once the installation has been performed successfully on all nodes of the farm, Tank takes care of publishing a VO specific software tag in the information system.

The management of concurrent installations for the same VO is also performed by Tank. If an installation or upgrade process is going on, the system stops another installation process from the same VO because of a temporary lock that lasts until the process ends. Tank controls the installation/removal process for a specific WN by setting an appropriate field in the database. In this way the installation on WNs takes place only if allowed. Configurations with or without a shared file system or mixed are therefore supported.

## 4.1. The Implementation

Tank&Spark has been entirely written in C++. The server exposes its methods through the SOAP protocol using gSOAP v2.6.

The Tank server uses the CERN implementation of the GSI plug-ins for gSOAP. Via a local grid-mapfile or other mechanisms (such as the EDG LCAS server plug-ins [15]) that maps specific software management roles to local accounts, only authorized users (or users with the right role) are allowed to perform installation tasks.

However, a module to interface the service to a generic high-level security interface described in [16] is already foreseen. In this way the server can dynamically support multiple authentication mechanisms.

**4.1.1. The Tank Database Tables.** The MySQL database on the server side keeps track of information regarding the status of the system. There are six main tables.

The *Flag*s table manages the propagation of the software installation to all the nodes once the software has been successfully installed by Spark and the new software tag has been registered in this table. This table contains 8 fields: the name of the software package installed, the status of the installation process (installation in progress, removal in progress, installation OK, removal OK, generic failure), the local account used to perform installation, time of installation or removal, e-mail of the ESM, a unique process identifier used as a key, a counter that is incremented everytime the installation/removal has been performed with success on a WN.

The *Host* table contains information about the status of the WNs: name and IP address of the WN, status (off if the node was not reachable for more than half an hour), last time that an interaction with the WN took place, type (if it shares the filesystem with other WNs or not – the default for all nodes is specified in the service configuration file), addition and removal time for this WN in the system. Through this table and through the configuration of the cron entries running on the WNs, the site administrators can control when an installation takes place.

The *Monitors* table is used for reserving and authorizing installation time slots, avoiding clashes with concurrent installations for the same VO. This table also manages installations in shared areas on a WN, avoiding that the same installation takes places on other nodes sharing the same area. It stores the following information: the local user performing the installation, the VO name, the VO status to allow site administrators to describe time slots for a VO installation to take place, the "shared" status allows the service to determine if an installation in a shared area can take place.

The *Success* and *Failure* tables keep track of the successful and failed operations per installation process. They store the unique installation process identifier and the hostname of the specific WN. In Failure also a message containing the cause of the failure is stored.

**4.1.2. Other Implementation Details.** The MySQL back-end database can be easily replaced by other back-end databases such as Oracle.

The service uses r-sync to synchronize software directories. The r-sync mechanism is a plug-in and can be replaced by other tools.

A Tank server can serve multiple VOs, while at the moment, on each WN one crontab entry is needed per VO supported running under one of the ESM local accounts.

The authentication is now based on GSI. However, the interface to the high level interface developed by the LCG EIS team and described in [16] is foreseen to allow for other authorization mechanisms.

Tank&Spark is also interfaced to the gssklogd service used for the conversion of GSI credentials into AFS Kerberos tokens. This allows for installations in AFS served areas.

The toolkit is maintained using the GNU Autotools and distributed via RPMs.

**4.1.3. Tank&Spark Features.** Installation, configuration and maintenance are quite easy tasks as reported by the site administrators who have activated the service.

Both server and client are resilient to failures. If the server goes down while an installation request is on going, the user is notified and the installation is attempted later on. Nodes contacting the server will just retry at a later time.

If a WN goes down and looses the software disk, the server will take care of the situation triggering the installation of all missing VO specific software versions.

Tank&Spark can also detect NFS failures and stop installation processes that use specific NFS mounted areas. It can also notify site administrators about NFS stale mounts. As stated previously it provides also support for AFS shared areas.

Through Tank&Spark it is possible to track down software management actions, identify the ESM who has triggered them, and notify site administrators.

Tank&Spark can also coexist and operate in parallel with the LCG application software installation and management service. It allows for installation of multiple releases of the same software.

Special wrapper tools available on the UI allow an ESM to trigger the installation, removal or update of a specific software release on all sites supporting a given VO. This is done contacting the LCG Information System to retrieve the list of sites supporting a specific VO. Automatic failure management is being added.

## 5. Experimental Results

We performed some preliminary functionality and performances tests using the LCG Grid farm in Pisa and Legnaro (Tier-2 centers), Italy. The goal of the tests was to verify the full compatibility of the service with the LCG application software installation and management service, its full functionality, stability, and scalability. Pisa provides 10 WNs Dual PIII 1GHz with 512MB of RAM, Dual AMD 1.6GHz and 1GB of RAM and Dual Xeon 2.4GHz with 512MB of RAM. They are connected via FastEthernet. Tank was installed on the CE (a PIII 1GHz with 512MB RAM), while the software repository has been configured on the SE, a PIV 1.5GHz with 256MB.

Legnaro provides a farm of WNs with 70 Dual Xeon processors with 2.4 to 3.0GHz and 2GB RAM. The CE and SE have a similar configuration and the farm is connected to a Gigabit Ethernet switch.

After 25 days of running under heavy tests the Tank server has shown no problems with memory usage of about 4MB.

While in Pisa we simulated a mixed configuration with some WNs sharing a file system for experiment software and some with a local installation, in Legnaro all WNs shared an AFS served area. In both cases the sites were used by production Data Challenges jobs that were not affected by the tests performed. Both sites support 5 VOs. Even though the WNs in Legnaro shared an AFS served area, Tank&Spark was configured to perform local installations. 1.4 GB of application software for the Geant4 VO was installed and propagated. Because of the shared area, no files have been actually propagated after the first installation to all WNs, but the r-sync service has been quite stressed since it has performed a checksum of 1.4 GB of data for 70 different nodes, managing simultaneous requests.

In addition, we measured the CPU and memory consumption of the r-sync server process while executing the synchronization of the software directory of one single WN with the central software repository on the SE. In average, with the machines under test, the CPU load was of about 10-15% while the memory usage was about 6.5MB.

As far as the Tank server performance goes, on the Pisa site we monitored the CPU load of the server program lcg-utank receiving requests for each VO every 5 minutes. In our setup (2 VO and 10 WNs) the server receives 20 connections and the CPU load is negligible (0.1%).

## 6. Related Work

The Pacman software [9] has been developed by the ATLAS collaboration for software distribution, installation and configuration. Even though the package is very effective and allows for dependency management, it cannot be considered an alternative to Tank&Spark. In fact, it does not provide a service to trigger installation on a set of WNs. However, Pacman could complement our solution allowing for the management of software dependencies and roll-back features.

A very recent work on application software installation is the one being developed in EGEE: the gLite PackMan [17]. As presented this tool can be an alternative to Tank&Spark. However, PackMan does not offer a framework for ESMs, but forces the packager to give a very detailed description of the software via specific metadata files. In addition, PackMan does not tackle the problem of automatic installation on a farm of WNs.

## 7. Conclusions

In this work we have presented our experience with application software installation and management services, the list of requirements for an adequate tool, and our proposed solution Tank&Spark. Preliminary results show that the service proposed seems to be quite stable and performing. Further enhancements include a more reliable way to notify ESMs of the status of the installation process (now only e-mail is supported), a more efficient way to handle requests coming from a UI (avoiding the use of the VO Box), the possibility of rolling back to the last functional software installation if problems arises, native support for software dependencies, disk quota management, development of a real super-service for central software installation and management on the Grid, more large scale tests to better understand the scalability of the system in terms of local network traffic (propagation of the software on all WNs).

## References

[1] LHC Computing Grid Project: http://www.cern.ch/lcg
[2] Globus Project: http://www.globus.org
[3] Condor Project: http://www.cs.wisc.edu/condor/
[4] Virtual Data Toolkit: http://www.cs.wisc.edu/vdt
[5] European Data Grid: http://www.cern.ch/edg
[6] gLite Toolkit: http://glite.web.cern.ch/glite/
[7] GSI: http://www.globus.org/security
[8] GLUE: http://infnforge.cnaf.infn.it/glueinfomodel/
[9] Pacman: http://physics.bu.edu/%7Eyoussef/pacman/
[10] Natalia Ratnikova, Andrea Sciaba, Stephan Wynhoff, Distributing applications in distributed computing environment, VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research, June 24-28, 2002.
[11] Preliminary observations on LCG-2 based on the 2004 LHC data Challenges: LCG-GAG-DC04.
[12] Software Installation Requirement document, 2 Dec 2004:http://project-lcg-gag.web.cern.ch/project-lcg-gag/LCG_GAG_Docs/SoftInst.pdf
[13] Experiment Software installation on LCG-1, 7 Nov 2004: http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/SoftwareInstallation/index.html
[14] Private communication with David Groep, NIKHEF.
[15] Martijn Steenbakkers, Guide to LCAS version 1.1.16, 15 September 2003, http://hep-proj-grid-fabric.web.cern.ch/hep-proj-grid-fabric/documentation/lcas.pdf
[16] Simone Campana, Antonio Delgado Peris, Flavia Donno, Patricia Mendez Lorenzo, Roberto Santinelli, Andrea Sciaba', Toward a Grid Technology Indipendent Programming Interface for HEP Applications, CHEP 2004, Interlaken, Switzerland.
[17] gLite PackMan: http://agenda.cern.ch/askArchive.php?base=agenda&categ=a043837&id=a043837s1t0/transparencies