Grid Infrastructure Architecture
A modular approach from CoreGRID

Augusto Ciuffoletti[1], Antonio Congiusta[2], Gracjan Jankowski[3], MichałJankowski[3], Ondrej Krajiček[4], and Norbert Meyer[3]

[1] INFN/CNAF, Bologna, Italy
augusto@di.unipi.it
[2] DEIS, Università della Calabria, Italy
acongiusta@deis.unical.it
[3] Computer and Automation Research Institute, Poznan, Hungary
[gracjan,jankowsk,norbert]@man.poznan.pl
[4] Masaryk University, Czech Republic, Brno
krajicek@ics.muni.cz

Abstract. The European Union CoreGRID project aims at encouraging collaboration among european research institutes. One target of such project is the design of an innovative Grid Infrastructure architecture, specifically addressing two challenging aspects of such entity: scalability and security. This paper outlines the results of such activity, ideally extending the content of the official deliverable document.

Key words: Grid Computing, Grid Information Service, Workflow Management, Checkpointing, Network Monitoring

1 Introduction

According to [Foster et al., 2002], a Grid is a complex architecture consisting of a collection of resources, which are made available at user level through a number of services. Such definition opens the way to a number of *functional components*, whose definition is of paramount importance for the design of a Grid: their semantics anticipate the capability of a Grid to make an efficient use of the resources it contains, to offer differentiated levels of quality of service, and, in essence, to meet user needs. Given the complexity and importance of such infrastructure, its design should address modularity as a primary feature: services provided by the Grid infrastructure should be precisely defined as for their interface and semantics, and form an integrated architecture which is a framework for their implementation. Modularity makes viable the independent evolution of each component, and allows the customization of the overall infrastructure.

In order to guarantee interoperability among components, *standard* interfaces are not sufficient. In fact, the capabilities of a certain functional component should be well understood, and agreed in the community that develops other interoperating services: typical requirements address resource access, workflow management, and security. Such semantics should be compatible with the expected needs of the user, be it a human or a Grid-aware application.

In addition, past experiences [Laure et al., 2006] prove that there is a tradeoff between portability and reuse of legacy tools: when functionalities that were not designed for integration are included into an existing project, the whole project tends to inherit all portability problems of the legacy parts. A *plugin oriented* approach does not solve the problem, but tends to complicate the design, and may even restrict portability.

Taking into account such problems, we indicate a *wrapper oriented* approach: legacy tools are not directly included in the design, but accessible through interfaces that comply with portability requirements of the hosting environment. The agent that implements such functionality (the "wrapper") is in charge of publishing portability issues that characterize the specific resource.

One key issue in the design of a Grid environment is the technology used to support the Grid Information System (GIS). It is more and more evident that a unique technology (for instance, a relational database) cannot satisfy all needs, and may exhibit real scalability limits in case of take off of the Grid technology [BerkeleyDB,]. Here we propose a differentiated strategy for such vital component, splitting its functionality into a directory service, and a streaming support. The monitoring infrastructure provides input to the GIS: we describe such infrastructure decomposed into resource and middleware monitoring, workflow monitoring and network monitoring.

Another key aspect of a Grid infrastructure is job submission. According to the GGF guidelines in [Rajic et al., 2004], we consider a unique component that performs batch submissions, scheduling and local queuing, workload monitoring and control. However, such component needs support for checkpointing and accounting, two activities that appear to require capabilities that need to be addressed specifically. We introduce two components that implement such functionalities.

The resulting Grid infrastructure should address both the need of e-science applications, mostly oriented to storage and computation intensive applications with moderate scalability, and emerging industrial applications, where the demand is variegated and includes the management of a large number of small jobs: in this perspective, flexibility is mandatory to allow customization.

Since we want to follow a clean design strategy, we address interoperation and integration issues since the early steps, using the GIS as a backbone. As a consequence, the adoption of a programming style and tools that support polymorphism is mandatory: the "wrapper oriented" approach indicated above helps on this way.

In Section 2 we indentify the functional components, and in section 3 we consider a GIS which provides an integration backbone. In figure 1 we depict a schematic view of our proposal.

2 Functional components of a framework architecture

The focus of a Grid infrastructure is on resource management: the goal is to compose the operation of basic services into higher level tasks. To this purpose, the Grid infrastructure accepts and processes task descriptions that articulate a stepwise composition of computing activities. The use of appropriate basic services, whose availability is constantly monitored by a *Resource Monitoring* component, is scheduled after unfolding the dependencies between atomic computational tasks. Resource scheduling extends

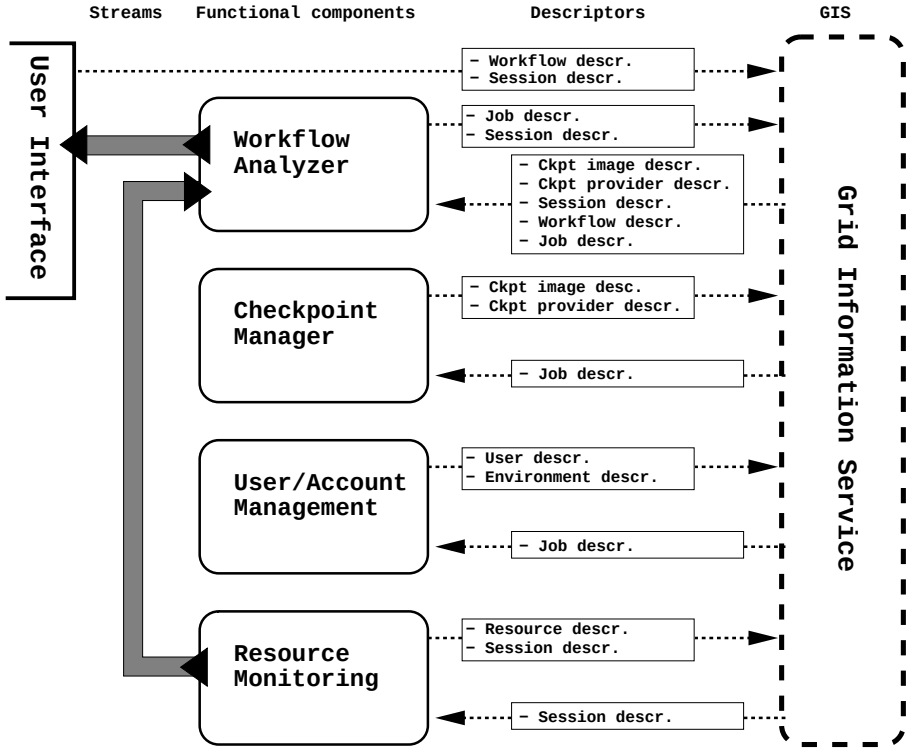

Fig. 1: Integration between the functional components of our framework. Each *component* is a distributed entity that contributes to resource management exchanging *descriptors* with other components. Persistent information flows are encapsulated into *streams*, represented by *session* descriptors)

not only in the name space, to determine which resource is to be used, but also in time, describing when a certain resource will be busy on a certain task.

The operation of assembling resources in order to perform a complex task is associated to the *Workflow Analyzer* component, whose role is to accept the operational description of a complex task, to manage, and to monitor its unfolding. The unfolding of a workflow must be sufficiently flexible, in order to cope with unanticipated events that may affect resources, either improving or degrading their performance. The appropriate way to cope with such events is the logistic re-organization of workflow execution, which usually entails the displacement of stateful computations, by re-instantiating services whose state corresponds to an intermediate computational step.

Two basic functionalities are offered: the registration of a snapshot of an intermediate state of a service, and the re-instantiation of the same service with the given intermediate state. All resources in a workflow participate to such reorganization, and the resulting workflow execution must be consistent with the expected semantics. The *Checkpoint Manager* component is in charge of supporting the logistic re-organization

of a workflow, preserving the relevant state information of a component services in preparation for the reconfiguration of the supporting low level services. Specific checkpointing indications are inserted in the operational description provided to the Workflow Analyzer.

Since resources are similar to goods, their sharing must be controlled accordingly, taking into account property and commercial value. In that sense, the Grid infrastructure provides identities to Grid users, and defines service semantics according to the identity of the user, thus enforcing individual property. Using the same tools, the usage of a certain service is quantified, and a commercial value associated with it. The *User and Account Management* component is appointed with such aspects.

The whole Grid infrastructure hinges upon the *Grid Information System* (GIS), which supports the integration between the parts of this distributed entity. From an abstract point of view, the content of the Grid Information System represents the state of the Grid, and is therefore dynamic. However, while some data remains constant for long periods of time, other are updated frequently, for instance when such information represents the residual (or preemptable) share of a resource.

The activity of a component is pervasive, and many distinct agents contribute to its implementation: for instance, each site can provide a Workflow Analyzer agent in charge of accepting user requests. Such approach fits naturally with security requirements, which are based on mutual identification among agents.

Here we give a summary of the functionalities each component offers, and we outline their internal structures: we use as a reference the work of the partners of the CoreGRID Institute on Grid Information, Resource and Workflow Monitoring.

2.1 Workflow Analyzer

The Workflow Analyzer cares about workflows management under several aspects such as mapping, scheduling, and orchestration of workflow tasks against the available, dynamic Grid resources. To such purpose, it has close interaction with the Grid Information System in order to discover and allocate appropriate resources. But, at the same time, it is also a source of information coming from the monitoring of the workflows being executed: most of such information is reused by the Workflow Analyzer itself for adjusting the ongoing executions.

A Grid workflow can be specified at different levels of abstraction: in [Deelman et al., 2003] *abstract workflows* and *concrete workflows* are distinguished, the difference being whether resources are specified through logical files and logical component names or through specific executables and fully qualified resources or services. According to this approach the workflow definition is decoupled from the underlying Grid configuration.

For this reason, the mapping phase (also referred to as matchmaking) is particularly important for selecting the most suitable resources that better satisfy the constraints and requirements specified in the abstract workflow, also with regard to quality of service and workload. The mapping process produces a concrete workflow that is suitable to be executed by a workflow engine, providing for scheduling and execution management capabilities. It is worth observing that, in case of dynamic scheduling, it is possible to re-invoke the mapping process at runtime, in order to modify the concrete workflow instance as a result of relevant events modifying the status of candidate resources.

However, it is important to instrument job descriptions before actual execution, in order to ensure that the workflow execution is suitably checkpointed: succinct requirements about workflow recoverability are in the Workflow description provided by the user.

When the workflow enters the running state, the Workflow Analyzer monitors its advancement, and takes appropriate actions in response to relevant events. During the workflow execution, monitoring is essentially related to the observation of the workflow status. In particular, information about the execution of each single job included in the overall workflow is reported by the monitoring system. Typical information is constituted by services execution status, failures, progress, performance metrics, etc.

In case of failure, the workflow execution service itself tries to recover the execution, for example by reassigning the work to a different host in case of host failure. To implement fault tolerance on a more refined extent, it is necessary whenever possible to trigger checkpoint recording, and drive the restart of one or more jobs from the last available checkpoint. The decision whether to checkpoint or restart a workflow is made on the basis of information from Resource monitoring.

2.2 Checkpointing

The Checkpointing component is built around the idea of Grid Checkpointing Architecture [Jankowski et al., 2006, Jankowski et al., 2005], a novel concept that defines Grid embedded agents and associated design patters that allows the integration of a variety of existing and future low-level checkpointing packages.

The emphasis has been put to make the GCA able to be integrated with other components and especially with the upper layer management services, for instance the Grid Broker or the Workflow Analizer. The main idea of the GCA boils down to provide a number of Checkpoint Translation Services (CTS) which are treated as drivers to the individual low-level checkpointing packages. The CTSes provide a uniform front-end to the upper layers of the GCA, and are customized to the underlying low-level checkpointers.

When the CTS is deployed on a Computing Resource, the GCA has to be informed about its existence. To fulfill this requirement each CTS is registered at component named Grid Checkpointing Service (GCS), and the GCS further exports the information about the available CTSes and related properties to the GIS, so that the GIS becomes the mechanism that connects the GCA with the external Grid environment. Additionally, from the point of view of the Workflow Analyzer, the GCS is the gateway to which a checkpoint request has to be sent. When the GCS receives the request of taking the checkpoint of a given job, it forwards the request to the appropriate CTS. The GCS is able to find the adequate CTS using the information that the execute-job-wrapper registers when a checkpointable job is started.

The execute-job wrapper is a special program provided together with an associated CTS. The component that is in charge of submitting the user's job to the given Execution Manager replaces the actual job with the adequate execute-job wrapper and passes to the second the original job and the global identifier assigned to this job. Which execute-job wrapper is to be used depends on which CTS has been matched to the job's requirements, according with GIS records. When the execute-job wrapper is started it

appropriately configures the GCA environment and finally with help of exec() syscall replaces itself into the original job.

When the Workflow Analyzer decides that a given job is to be recovered, then the job has to be resubmitted in an adequate way: one relevant issue is that the job can be resubmitted only to a Computing Element associated with a CTS of the same type that was used to checkpoint the job. When a proper Computing Element is found the job is resubmitted to it, but instead of resubmitting the original job itself the recovery-job-wrapper is resubmitted. The original job, as well as the identifier of the checkpoint that is to be used in the recovery process, are passed to the recovery-job-wrapper as the arguments.

The recovery-job wrapper is the counterpart of the execute-job wrapper used for the recovery activity. The recovery-job wrapper starts fetching the image, and the subsequent actions are similar to those performed by the execute-job wrapper. As a last step, the recovery-job wrapper calls the appropriate low-level checkpointing package to recover the job using the image indicated by the calling Workflow Analyzer.

The GCA shares the motivations of the Grid Checkpoint and Recovery working group of the GGF [Stone et al., 2005], which is to include the checkpointing technology into the Grid environment. However, the GCA focuses mainly on legacy checkpointing packages and, notably those that are not Grid-aware, while the GridCPR ”is defining a user-level API and associated layer of services that will permit checkpointed jobs to be recovered and continued on the same or on remote Grid resources”. Therefore, while GridCPR works on a specification that future Grid applications will have to adhere to in order to make them checkpointable, our effort is towards the integration of existing products into a complex framework.

2.3 User and Account Management

The User and Account Management component [Denemark et al., 2005] offers a controlled, secure access to grid resources, complemented with the possibility of gathering data from resource providers in order to log user activity for accounting and auditing purposes. These objectives are realized introducing authorization, ensuring an appropriate level of job isolation and processing logging data. A *virtual environment* encapsulates jobs of a given user and grants a limited set of privileges. Job activity is bound to a user identity, which is qualified as a member of an organization.

The User and Account Management component is a pluggable framework, that allows combining different authorization methods (e.g. gridmap file, banned user list, VO membership based authorization) and different implementations of environments (virtual accounts, virtual machines, and sandboxes). The configuration of the framework is quite flexible and depends on detailed requirements which may vary between the resources, so the administrators may tune local authorization policy to the real needs and abilities.

The internal architecture of an agent consists of 3 modules: an authorization module, a virtual environment module and a virtual workspace database. The authorization module performs authentication first (based on existing software, such as Globus GSI). The authorization is done by querying a configurable set of authorization plugins. The virtual environment module is responsible for creation, deletion and communication with

virtual environments modeled as Stateful Resources. The module is also pluggable, so it is possible to use different implementations of VE. The database records operations on the virtual environments (time of creation and destruction, users mapped to the environment, etc.). These records together with the standard system logs and accounting data, provides complete information on user actions and resource usage.

2.4 Resource Monitoring

The information on resources and accompanying middleware is provided by the Resource Monitor. Resource Monitor component collects data from various monitoring tools available on the grid. We do not presume any particular monitoring approach, since the current state of the art provides quite wide range of monitoring toolkits and approaches. It is however a difficult task to integrate and process monitoring information from various monitoring tools. Moreover, we cannot assume any scale of the resulting infrastructure thus scalability of the proposed solution, both in terms of amount of monitored resources and required processing throughput for monitoring data, must be emphasized.

To achieve the desired level of scalability, with security and flexibility in mind, we propose the design of a Resource Monitor based on the C-GMA [Krajicek et al., 2006] monitoring architecture. C-GMA is a direct extension of the GMA [Tierney et al., 2002] specification supported by the Open Grid Forum.

The key feature supplied by the C-GMA is the introduction of several metadata layers associated with services, resources and monitoring data. The metadata may specify the data definition language used by the services, the non-functional properties and requirements imposed by the services and resources (such as security and QoS-related requirements) and others. The metadata are used in the *matchmaking* process implemented by the C-GMA architecture, which is essentially a reasoning on provided metadata about the compatibility of the services and data described by them. When the examined parties are considered compatible, the “proposal” is sent to them to initiate a potential communication. In this way, and with the introduction of various translation components, the C-GMA architecture enables the exchange of monitoring data between various monitoring services.

The Resource Monitor service leverages this functionality by connecting to the C-GMA monitoring architecture and using translation services for various monitoring toolkits it collects the monitoring data and supplies them to the Grid Information System.

Special attention is paid to Network Monitoring, since scalability issues appear as challenging. We have identified one basic agent, the Network Monitoring Element, which is responsible of implementing the Network Monitoring Service [Ciuffoletti and Polychronakis, 2006]. Network Monitoring Elements (NMEs) cooperate in order to implement the Network Monitoring component, using mainly lightweight passive monitoring techniques. The basic semantic object is Network Monitoring Session, which consists in the measurement of certain traffic characteristics between the Domains whose NMEs participate in the session.

To improve the scalability of the Network Monitoring Service, the NMEs apply an overlay Domain partition to the network, thus decoupling the intra-domain network

infrastructure (under control of the peripheral administration), from the inter-domain infrastructure (meant to be out of control of the peripheral administration). According to the overlay domain partitioning network, monitoring sessions are associated to Resources denoted as Network Elements (NE), corresponding to inter-domain traffic classes.

The overlay domain partition is maintained in an internal distributed database, which allows the coordination among Network Monitoring Elements. The management of network monitoring sessions includes the control of periodic sessions, as configured by network administrators, and of on-demand sessions dynamically configured by applications, and uses a scalable peer to peer mechanism to diffuse updates.

3 Integration between functional components

The central idea of the proposed architecture is to convey all the data through the Grid Information Service in order to have a standard interface across the different administrative sites and services (see [Aiftimiei et al., 2006, Andreozzi et al., 2005] for a similar approach).

One relevant feature of a data repository, and of the Grid Information System, is the volatility of its content. At one end we find "write once" data, that are not subject to update operations and have a relatively low volatility. At the other hand we find data that are frequently updated. The functionality associated to the Grid Information System is a mix of both: while certain data, like a Workflow description, fall in the "write-once" category, other kind of data, like resource usage statistics, fall into the category of data that are frequently updated: a solution that devises a common treatment for both kinds of data suffers of a number of inefficiencies, first the lack of scalability.

Therefore our first step is to recognize the need of distinct solutions for persistent and for volatile data. One criteria to distinguish the two kinds of data is the length of the time interval during which the information remains unchanged, under normal conditions. Here we assume that a significant threshold is given by the typical job execution time: we consider as persistent those pieces of information that, under normal conditions, remain valid during the execution of a job. We call such informations *descriptors*: starting from the specifications of the components that compose our framework given in previous sections, we now classify the descriptors that are exchanged among them, and that collectively represent the persistent content of the Grid Information System.

Workflow descriptor It is acquired from a user interface by the Workflow Analizer component. It has the function of indicating the stepwise organization of a Grid computation. It contains high level indications about the processing requested at each step, as well as dependencies among individual steps. It should be designed in order to hide all unnecessary details, for instance package names or versions, and focus on the functionality (for instance, "fast fourier transform", or "MPEG4 compression"). During workflow execution, such structure is used by the Workflow Analizer component in order to monitor workflow execution.

Job descriptor It is produced by the Workflow Analizer component, and fed to various other components: it is used by the Checkpointing component in order to prepare

the execution environment with checkpointing facilities, and by the User and Account Management component in order to associate an appropriate environment to its execution. The Job description is used by the Workflow Analizer component in order to instruct resources about their activity, and during workflow execution, to monitor workflow advancement.

Checkpoint Image Descriptor It is produced by the Checkpointing component (in case of the GCA, this descriptor is produced by the CTS) upon recording a new checkpoint. The descriptor contains the bookkeeping data regarding the newly created image. The data can be used by the Workflow Analizer in order to find the identifier of the image that is to be used in order to perform recovery and migration. The GCA itself, basing on the descriptor, is able to fetch the image to the node on which the given job is to be recovered.

Checkpoint Provider Descriptor It is produced by the Checkpointing component. The descriptor advertises the location of service that provides access and unified interface to a particular low-level checkpointing package. The Workflow Analizer uses such descriptior to find the node that provides the desired checkpointing package, as specified in job descriptor. Upon recovery, the descriptor allows finding the nodes offering the same package used for checkpointing.

Session descriptor It is produced by a generic component, and supports the exchange of volatile data, as described below.

User descriptor It is produced and used by the User and Account Management component. It contains a description of a user, like its name, institution, reachability, role, as well as security related data, like public keys. The Workflow Analysis component uses such data to enforce access restrictions when scheduling a Workflow.

Environment descriptor It is produced and used by the User and Account Management component. It contains references to he descriptions of the resources associated to a given processing environment, as well as the access modes for such resources. This may correspond, for instance, to what is needed to run a specific kind of job, and to the identities of the users that are allowed to operate within such environment. The Workflow Analysis component uses such data in order to process a workflow description.

Resource descriptor It represents usual resource descriptions, including storage, processing, network and network monitoring elements. The identification of a resource includes its network monitoring domain. The Workload Analyzer uses such descriptions in order to schedule job execution, and allocate checkpoint storage.

The management of *descriptors* relies on a directory-like structure. Such structure cannot be concentrated in replicated servers, but distributed in the whole system based on local needs. Functional components that need to have access to such data should address a proxy, which makes available the requested information, or add/delete a descriptor. An LDAP directory provides a first approximation of such entity: however, descriptors are not organized hierarchically. A better alternative is an adaptive caching of those descriptors that are considered locally relevant: for instance, the descriptor of a monitoring session might be cached in a GIS proxy near the monitored resource. Descriptors are diffused in the system using a low footprint, low performance broadcast protocol, and cached wherever needed.

The volatile data is represented by data that change during the execution of a job: a typical example is the workload of a computing element. Such data are produced by one of the components described in the previous section, and made available to a restricted number of other components. The storage into a globally accessible facility, included a distributed relational database, seems inappropriate since the information is usually transferred from a source to a limited number of destinations. The concept that is usually applied to solve such kind of problems is the multicast.

A multicast facility appropriate for diffusing volatile data of a Grid Information System has many points in common with a Voice over IP infrastructure: the container of the communication is similar to a Session (as defined in the SIP protocol). In contrast with a typical VoIP application, the data trasfer within a session in mainly uni-directional and requires a low bandwidth with moderate real time requirements: we call *streams* the information flows associated to the trasport of volatile data within a Grid Information System.

All of the components outlined in section 2 are able to initiate or accept a session with another component: security issues are coped with using the descriptors associated with the agents. E.g., a Resource will accept a call only from a Workflow analyzer that submitted a job. Here we outline some of the relevant streams:

Resource usage stream It is originated by a resource, like a Storage Element, and summarizes the performance of the resource, as well as the available share of it. Typical callers are the Workflow Analyzer, either during the resource selection or the execution phase.

Workflow advancement stream It is originated by a Workflow Analyzer component, and reports the caller about the workflow advancement. Typical callers are user oriented interfaces.

One characteristic of a session, that makes it not interchangeable with a directory service, is that the establishment of a session has a relevant cost, which is amortized only if the session persists for a significant time interval. For this reason we include sessions in the number of entities that have a descriptor recorded in the Grid Information Service.

Such descriptor advertizes the existence of a given session: it is a task of the callee to create and make available an appropriate *Session descriptor*, as outlined above. Sessions can be activated on demand, or be permanently available: such option depends on the balance between the workload needed to activate a new session on demand, and of keeping it warm for connection. E.g., Network Monitoring sessions will be mostly activated *on demand*, while Storage usage statistics can be maintained permanently active.

4 Comparison with other works

The architecture we propose takes into account the goals and achievements of a number of scientific, as well as industrial projects that accepted the challenges proposed by the design of an effective grid infrastructure.

One outstanding project which is being developed to meet the requirements the scientific community is gLite: it is developed within the European EGEE project, the

successor of DATAGRID. Its purpose is to capitalize tools and experience matured in the course of DATAGRID, in order to assemble a Grid infrastructure usable for high performance computation, first the LHC experiment on schedule for the next year.

We consider gLite [Laure et al., 2006] as a precious source of experience about a real scale Grid environment. We considered as relevants the inclusion of a number of features that are not considered, or considered at an embrional level, in gLite. Namely, we introduce a specific component that takes into account job checkpointing, we adopt a more powerful workflow description language (but gLite is working towards a DRMAA [Rajic et al., 2004] compliant interface), we take into account the task of workflow monitoring under scalability requirements, also considering networking resources, we differentiate the functionality of the GIS into a high latency directory service, and a multicast real-time streaming service. Overall, with respect to gLite, we considered the need for a wide portability: although such problem is not overly relevant for the environment for which gLite has been developed, we considered it relevant in a broader scope. To improve portability we suggest the realization of an integrated framework for the whole infrastructure, hosting legacy components encapsulated in specific wrappers.

With respect to implementations based on the DRMAA proposed standard [Rajic et al., 2004] we consider the interactions between Resource Management and Checkpointing, since we observe that the resource management is the component in charge of instructing the resource about activities relevant to recovery and relocation of running jobs. Therefore we describe an interface between a component in charge of managing a transparent management of checkpoints, and another in charge of interpreting user requests.

The N1GE by Sun [Bulhes et al., 2004] is considered as a relevant representative of the industrial effort towards the implementation of a Grid infrastructure. Such project recognises the problems arising from the adoption of a monolythic relational database, and adheres to the DRMAA standards as for job descriptions. In order to overcome scalability limits imposed by a monolythic databases, it adopts a more flexible commercial database, Berkeley DB [BerkeleyDB,]. In our proposal we identify the kind of services of interest for our infrastructure, and indicate complementary solutions, that cannot be assimilated to a relational database. This should improve scalability and resource usage.

The focus of the GPE [Ratering, 2005] prototype by Intel is to bridge users from non-Grid environments, and to provide an interface that will remain sufficiently stable in the future, shielding the user from the changes of a still evolving middleware technology. Therefore the focus is on the provision of a powerful interface that adapts to several kinds of users. In order to take advantage of legacy tools, like UNICORE [UNICORE, 2003], security issues are delegated to a specific component, the Security Gateway, that enfoces a secure access to sensitive resources. In our view this is a source of problems, since the presence of a bottleneck limits the performance of a system. Instead, we indicate a pervasive attention to security issues, in order to implement appropriate security issues inside each agent.

We pay special attention to a Grid resource that is often overlooked: the network infrastructure. Such resource is difficult to represent and to monitor since, unlike other resources, its complexity grows with the square of system size. Yet this resource has a vital role in a distributed system as a whole, since its availability determines its performance, and directly reflects on jobs performance.

5 Conclusions

CoreGRID is an European project whose primary goal is to foster collaboration among european organizations towards the definition of an advanced Grid architecture. One of the tasks that contributes to this achievement is targeted at the description of an *Integrated Framework for Resource and Workflow Monitoring*. In order to enforce integration since the early steps, the research and development activities from several research groups are included in the same container, with frequent and planned meetings.

This paper presents an early result on this way, after two years from the beginning of the project. We have tried to understand the problems left opened by other similar initiatives, specifically aiming at scalability and security issues, and identified the actors inside our framework. The research groups have produced relevant results for each of them that are only summarized in this paper; instead, we focus on the integration among such actors, based on *descriptors* advertised in the *Grid Information Service*.

Acknowledgements This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265)."

References

[Aiftimiei et al., 2006] Aiftimiei, C., Andreozzi, S., Cuscela, G., Bortoli, N. D., Donvito, G., Fantinel, S., Fattibene, E., Misurelli, G., Pierro, A., Rubini, G., and Tortone, G. (2006). GridICE: Requirements, architecture and experience of a monitoring tool for grid systems. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP2006)*, Mumbai - India.

[Andreozzi et al., 2005] Andreozzi, S., De Bortoli, N., Fantinel, S., Ghiselli, A., Rubini, G., Tortone, G., and Vistoli, C. (2005). GridICE: a monitoring service for Grid systems. *Future Generation Computer Systems Journal*, 21(4):559–571.

[BerkeleyDB,] BerkeleyDB. Diverse needs, database choices. Technical report, Sleepycat Software Inc.

[Bulhes et al., 2004] Bulhes, P. T., Byun, C., Castrapel, R., and Hassaine, O. (2004). N1 grid engine 6 – features and capabilities. Technical report, SUPerG.

[Ciuffoletti and Polychronakis, 2006] Ciuffoletti, A. and Polychronakis, M. (2006). Architecture of a network monitoring element. Technical Report TR-0033, CoreGRID.

[Deelman et al., 2003] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and Koranda, S. (2003). Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39.

[Denemark et al., 2005] Denemark, J., Jankowski, M., Matyska, L., Meyer, N., Ruda, M., and Wolniewicz, P. (2005). Usermanagement for virtual organizations. Technical Report TR-0012, CoreGRID.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed systems integration.

[Jankowski et al., 2006] Jankowski, G., Januszewski, R., Mikolajczak, R., and Kovacs, J. (2006). Grid checkpointing architecture - a revised proposal. Technical Report TR0036, CoreGRID - Network of Excellence.

[Jankowski et al., 2005] Jankowski, G., Kovacs, J., Meyer, N., Januszewski, R., and Mikolajczak, R. (2005). Towards Checkpointing Grid Architecture. In *PPAM2005 proceedings*.

[Krajicek et al., 2006] Krajicek, O., Ceccanti, A., Krenek, A., Matyska, L., and Ruda, M. (2006). Designing a distributed mediator for the C-GMA monitoring architecture. In *In Proc. of the DAPSYS 2006 Conference*, page to appear, Innsbruck.

[Laure et al., 2006] Laure, E., Fisher, S., Frohner, A., Grandi, C., Kunszt, P., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Hemmer, F., Meglio, A. D., and Edlund, A. (2006). Programming the grid with glite. Technical Report EGEE-TR-2006-001, EGEE.

[Rajic et al., 2004] Rajic, H., Brobst, R., Chan, W., Ferstl, F., Gardiner, J., Haas, A., Nitzberg, B., and Tollefsrud, J. (2004). Distributed resource management application API specification. Technical report, Global Grid Forum. http://www.ggf.org/documents/GWD-R/GFD-R.022.pdf.

[Ratering, 2005] Ratering, R. (2005). Grid programming environment (GPE) concepts. Technical report, Intel Corporation.

[Stone et al., 2005] Stone, N., Simmel, D., and Kielmann, T. (2005). An architecture for grid checkpoint and recovery (gridcpr) services and a gridcpr application programming interface. Technical report, Global Grid Forum. draft.

[Tierney et al., 2002] Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., and Wolski, R. (2002). A grid monitoring architecture. Technical Report GFD 1.7, Global Grid Forum.

[UNICORE, 2003] UNICORE (2003). UNICORE plus final report. Technical report, BMBF Project UNICORE Plus.