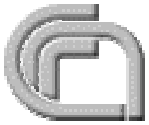
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

A model for a component based grid-aware scientific library service.

S. Lombardo, A. Machì

RT-ICAR-PA-04-01 **Gennaio 2004**



Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

A model for a component based grid-aware scientific library service

S. Lombardo, A. Machì

Rapporto Tecnico N.: **RT-ICAR-PA-04-01**

Data: **Gennaio 2004**

A model for a component based grid-aware scientific library service.

S. Lombardo[1], A. Machì [1]

[1]ICAR/CNR Department of Palermo
{s.lombardo, machi }@pa.icar.cnr.it

Abstract. The paper presents a hierarchical model for integrating structured HPC legacy software modules into a grid-adaptive scientific library service able to deliver quality computing-service on a performance contract basis. The service proactively administers both computing resources and configurable library software modules on behalf of client applications, and seamlessly supports their re-configuration according to grid-aware strategies of active Managers of client Applications. The library administrator logic mimics functionalities of components containers of service-oriented architectures. The authors discuss the coordination of library-serviced components into a grid-aware application developed according to the programming model being developed by the *Grid.it* Italian Grid Project. Sample architecture for the library service, based in *Grid.it* component technology, is sketched. The implementation level of its run time grid middleware is presented

1. Introduction

A scientific library is a set of structured code modules; developed to solve problems in a specific domain and it is used to build applications according to the Software Engineering reuse concept. This means that a module library should be robust, secure and easy to use. In specific domains, where high performance computing is needed, the library is often implemented by encapsulating parallel and distributed computation [1]. In particular, the structured parallel programming approach has embodied such knowledge into patterns for the management of set of processes described by notable Processes Graphs, called *skeletons* and *parmods* [2-3]. Skeletons are automatically coded by parallel compilers to keep high the parallel efficiency software portability, while maintaining low user parallel programming effort.
In the past decade code has been developed with such a structured approach for computing environments mapped on static networks of resources, and controlled by stable policies providing facilities for exclusive resource allocation. Such code, by itself, is inefficient on grid environments maintained as Virtual Organizations on wide-area networks of resources dynamically discoverable and shareable, because of the resource unreliability behaviour, intrinsic in the Virtual Organization model [4].
Discussion of methodologies and projects on self-adapting numerical software on the grid [5] is out the reach of the present paper. We just quote Netsolve and Ninf-G [6] projects aimed to provide library packages available on a grid-environment.

Netsolve is an easy-to-use tool to provide efficient and uniform access to a variety of scientific packages on heterogeneous platforms, based on RPC model which supplies load balancing, fault-tolerance and agent based grid-resource management.
Ninf-G provides a secure Remote Procedure Call invocation of library modules without any performance warranty.
Both these approaches make use of the Grid-RPC paradigm, a programming model based on client-server remote procedure calls on the Grid, and both supply client APIs to invoke library services. In Netsolve, a moderate degree of fault tolerance is maintained by an agent, attempting to find among the pool of available grid resources an appropriate server to optimally service client requests, keeping track of failed servers.
In the present paper we limit our perspective to easy grid enabling of scientific legacy software in a limited domain. From this restricted point of view main requirement for a grid-enabled library service is compliance, at resource level, with the checklist definition of a Grid system that: "coordinates resources that are not subject to centralized control", "using standard, open, general-purpose protocols and interfaces", "to deliver non trivial qualities of service" [7].
The first requirement implies allowing serviced library modules be orchestrated by client application. The second feature requires adoption of OGSA compliant interfaces, protocols and life cycle support service. Last requirement implies to implement a complex software engine for contract submission, violation detection (monitoring) and decision about reconfiguration of allocated resources.
To deliver non-trivial Quality of Service the library service should be able to honour performance contracts tailored to support both application biased and system biased optimisation strategies including:

1. **Real-Time**: execution of the library service within a specific range of time, with application self-management of grid-resources;
2. **Resource-Reservation**: execution on reserved specific nodes and connections.
3. **Priority**: best-effort pre-emptive on a pool of grid resources administered;
4. **Low cost**: cheapest service cost, low priority execution on free grid-resources;

In this scenario we propose a component-based model for implementing an HPC library service able to honour the described performance contracts. The required grid-awareness is distributed among various software elements of the environment playing different roles [8].
Section 2 introduces the reference component architecture. Section 3 sketches the distributed adaptivity model with entity and actors involved. Section 4 describes the integration of the adaptivity model with application management strategy. Section 5 describes the logical implementation of the library server prototype.

2. The *Grid.it* Component Model

Software component technology is a young programming-paradigm, even though its definition is quite old. Its aim is to enable the development of applications by composing existing software elements in an easy way. Several components models have been proposed. The technology used for implementing component-to-component

binding depends on required performance, interoperability and may change in according the scope of the connection. The Globus Project proposed the OGSA architecture for grid services and a component architecture for adaptive grid programming OGSA compliant has been defined in [9]

A new component architecture focusing on HPC grid programming is presently being developed by the Italian national *Grid.it* project [10]. *Grid.it* components are intended to support design of HPC applications over a grid middleware. The details of the architecture are a topic of current research. In the working model [11] components expose their functionalities through a series of interfaces that differ for the interaction paradigm:

- **RPC** interfaces conforming to the Remote Procedure Call standard model;
- **Event**: optional interfaces to receive and to send asynchronous events;
- **Stream**: optional unidirectional data-flow channels;
- **Configuration (active and introspection):** interfaces required by architecture for component configuration and status retrieval.

An application is modelled by composing compatible components in a graph that:

- may change arcs connecting components during the execution of the application;
- may have connections to components implemented according to other open standard component architectures (CCM, Grid Services…);
- includes a logic unit named *application manager* able to instantiate other application components, to connect them and (re) configure the application graph.

The *application manager* may be implemented as a single component or a coordination of components implementing the application coordination *strategy*.

3. A hierarchical component-based model for grid adaptivity.

As mentioned above, we suppose that grid-adaptivity can be modelled using different actors playing hierarchically cooperative roles [8]. These roles model may be mapped onto a component-based grid software infrastructure.

At top level stays the *active resource&execution manager*: its role involves (re) selection of proper resources (nodes and library). It maintains grid discovery ability, detailed grid-awareness, reservation privileges and an adequate policy to coordinate resource provision in order to ensure application performance (ability in performance contract negotiation).

At second level is the *proactive resource administrator:* this role requires definition of each application performance in terms of a performance contract, monitoring of performance and a policy for reconfiguration. It represents the front-end of the library-services and its goal is to monitor contracted performance and to adapt management of available grid resources for optimal execution of the library modules. This action takes advantage of self-optimisation capability embodied in parallel skeleton templates [12].

A *reactive quality-service coordinator* is already implemented in some parallel skeletons. His role is load (re) balancing of physical processes over a cluster of virtually privates inhomogeneous resources labelled with their effective quality

indexes plus partial reconfiguration of the processor graph after in the event of their variations.
Monitoring of resource status, support for application deployment, detection and registration of events requiring attention and possible adaptation may be performed by a *passive resource coordinator* staying at lowest level.
The *resource administrator* and the *resource coordinator* roles may be assigned to grid middleware while the quality-service coordinator role is assigned to skeletons. The *resource administrator* mimics functionalities of components containers of service-oriented architectures. The *active resource&execution manager* instead are external actors for the proposed service architecture and they may be elements of a Problem Solving Environment (PSE)[13], of a Grid-Portal or of a generic environment for grid programming. For example, the Application Manager of the Grid.it component-programming model could play the execution manager role.

4. Integration of the library server with Grid.it application strategy

Our proposed library server is a grid-aware application, which exposes to several external grid applications a service for library modules orchestration.

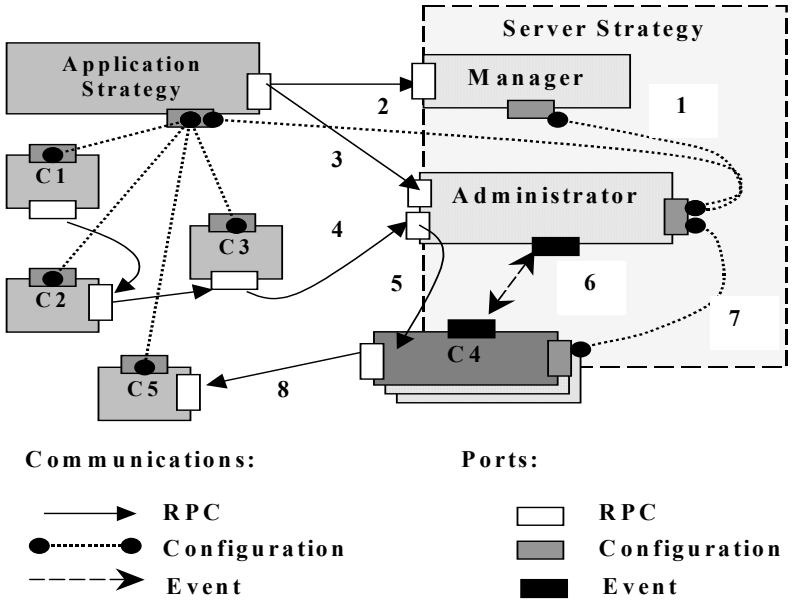


Fig. 1. Client Application interaction

Even if it uses the *Grid.it* component architectural model the library service provides an open standard OGSA-based interface to not limit client applications architecture. *Grid.it* supplies a rich and expressive model to design a hierarchical and distributed application management that is particularly useful for the implementation of our hierarchical roles.

Figure 1 shows a sample a scenario of interaction between the library server and a *Grid.it* client-application to implement a two-fold optimisation strategy.

In this scenario, RPC library module ports are not available to client applications because they are hidden by the *Administrator*, but library module uses RPC port could be connected with other components as established by the strategy of client application *Manager* during the factory operation. The *Administrator* schedules client service requests in a priority queue and processes them on the basis of their associated performance contract. According to the contract, an application biased or a global resource policy is selected when selecting the proper subset of pool resources to map the Virtual Process Graph of each request. Requests are served when the minimum required resource set is available.

Here is a description of the interaction sequence between client application and the library server:

- The *Resource Manager* selects a pool of appropriate resources (hosts of the VPG) and library modules (discovered from an external repository) and communicates them to the Component Administrator through the *configuration* port (1).
- The *Component Administrator* acquires the selected resources and updates its internal *register* to provide the available library modules.
- The *Resource Manager* negotiates, after investigating about the available libraries via the *introspection* port (2), a library module invocation via the Component Administrator *factory* port (3), eventually specifying bindings for module use ports. During this operation the client *Manager* submits a performance contract template as mentioned above. The administrator deploys (if not already done) the library model over each node in the pool independently from its effective configuration.
- When the library service is invoked (4) by a client *Application* component the *Administrator* schedules the request and, when the requests come to queue top, it dispatches the request to the *Library module* (5).
- Start of contract monitoring and adaptivity (re) configuration. Using the event bus provided by the run-time support, the *Library module* throws events (6) to communicate the progress status of its job (checkpoiting). It can also register itself to receive events regarding workflow modifications. The *Manager* monitors the respect of performance contracts via the *Adminstrator introspection* port. In case of violation it issues a (re) configuration command via the *Adminstrator configuration* port. The *Administrator*, in turn, throws proper (re) configuration events to *Library module*, which performs self-reconfiguration.

5. Architecture of the Library Server

In this section we give a description of the internal architecture of the library server prototype being developed. A prototype of the *Virtual Private Grid* (VPG) runtime support is in alpha-test stage. It provides an XML-based RPC interface with commands to hide management of physical nodes and processes on a grid environment. Grid nodes management and file transfer are implemented over the Globus toolkit V2.4 services: GRAM (for start-up of the Remote Engine), Grid-FTP for deploying library modules and GSI protocols for authorization and secure file transfer. Detailed design of the *Component Administrator* and *Application Component* prototypes is in course. VPG prototype is built over the ACE toolkit [14].

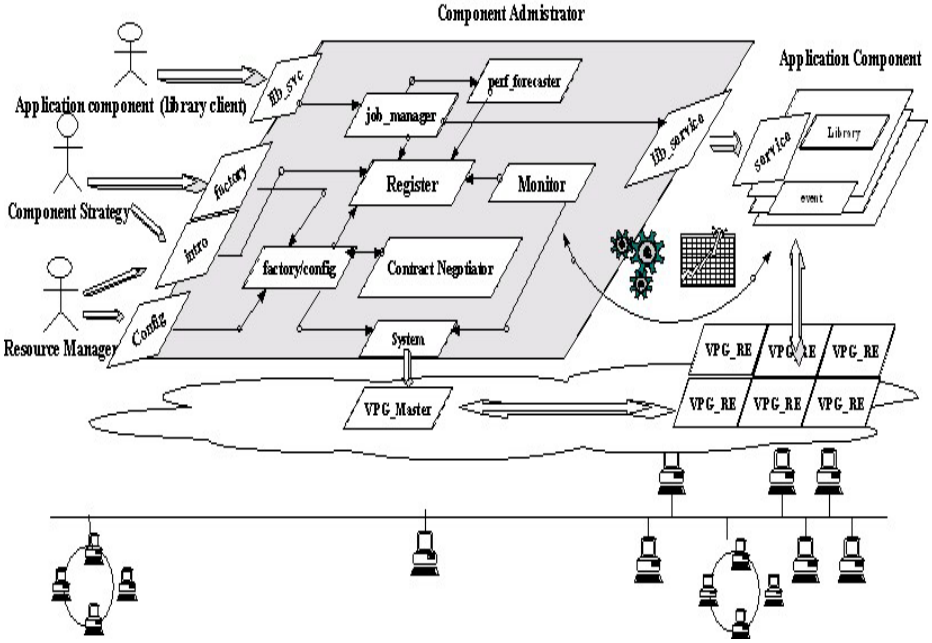


Fig. 2. Library server architecture

Figure 2 shows components organization: over the resource pool stays the VPG runtime support. On the VPG stay the Component Administrator and all deployed application components wrapping library modules. The Resource Manager component is actually an actor external to the library service.
A brief description of the components follows.
The **Component Administrator** implements proactive resource administrator role and it's modelled as a horizontal composition of simpler components: the *factory, register, monitor, performance-forecaster, contract negotiator* and the *job manager*. Its capabilities are exposed through the following RPC ports:

- **factory**: an extension of the factory design pattern [15-16] to the domain of distributed computing, similar to the OGSA Factory Port Type. Its methods allow to deploy a module library on the pool of managed grid resource (VPG);
- **lib-service:** supports invocation of library modules methods;

- **introspection**: supports queries about the status of installed or activated library modules as well and about VPG resource status.
- **config:** it supplies an high-level facility to mount/un-mount grid resources (hosts of the VPG) and library modules and supports late- (re) binding of their use ports;

The **Application** component embodies one library module and can encapsulate parallel and distributed computation. It plays the role of a *reactive quality-service coordinator* being able to react to events supplied by the VPG middleware. It can register itself for interesting (re) configuration events sent by the monitor and can be enabled to send events about its proceeding status. It exposes the following ports:

- **service:** the behaviour of the module (RPC interface use and/or provide), as we mentioned above the provide port is always bound to the component administrator although the use port could be bound whit other client application component;
- **event_sink:** allows the component to receive interesting events;
- **event_source:** allows the component to send events;

The **VPG-Master** implements the front-end of the *passive resource coordinator* middleware. It manages the life cycle of sets of processes on top of a pool of grid resources and offers to the upper layers a *Virtual Private Grid* facade similar to a standard processor cluster facility. It exposes the following ports:

- **service_provide**: exposes methods for administrating node facilities (mount, un-mount, keep alive, get-status) and for controlling set of processes (deploy, start, kill, delete, get-status);
- **event_sink**: allows the component to receive interesting events;
- **event_source**: allows the component to send events;

The **VPG-Remote Engine** is a daemon running on each host mounted on VPG as a slave for VPG-Master requests. It implements the remote run-time environment, and administers, under master control, local processes lifecycle. It provides to the application-component an event bus for meaningful events registration and notification, to enable its reactive role. Its ports are:

- **service_provide**: exposes capabilities for administrating a node of the VPG;
- **event_sink**: allows the component to receive interesting events;

References

[1] P. D'Ambra, M. Danelutto, D. di Serafino, M. Lapegna, Integrating MPI-based numerical software into an advanced parallel computing environment, in "Proc. of the 11th EUROMICRO Conf. on Parallel, Distributed and Network-based Processing", IEEE Pub., 2003, pp. 283-291.

[2] J. Darlington, Y. Guo, H. W. To, J. Yang, Parallel skeletons for structured composition, In Proc. of the 5th ACM/SIGPLAN Symposium on Principles and Practice of Paral1el Programming, Santa Barbara, California, July 1995, SIGPLAN Notices 30(8),19-28.G.

[3] M. Vanneschi: The programming model of ASSIST, an environment for parallel and distributed portable applications. Parallel Computing 28(12): 1709-1732 (2002).

[4] F.Berman, G.C. Fox, A.J.G.Hey: Grid Computing. Making the Global Infrastructure a Reality. Wiley 2003
[5] Zizhong Chen, Jack Dongarra, Piotr Luszczek, and Kenneth Roche"Self Adapting Software for Numerical Linear Algebra and LAPACK for Clusters www.cs.utk.edu/~luszczek/articles/lfc-parcomp.pdf [5] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, H. Casanova : "GridRPC: a Remote Procedure Call API for Grid Computing
[6] S. Agrawal, J Dongarra, K Seymour and S. Vadhiar "NetSolve: past,present and future – a look at a Grid enabled server" in Grid Computing: Making the grid infrastructure a reality.J. Wiley and Sons 2003.
[7] What is the Grid? A Three Point Checklist. I. Foster, GRIDToday, July 20, 2002
[8] A. Machì, S. Lombardo "A conceptual model for grid-adaptivity of HPC applications and its logical implementation with components technology" Accepted for presentation at ICSSA04/AGCPA Krakow, Poland June 2004.
[9] Rob Armstrom, Dennis Gannon, Katarezyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinsk. "Toward a common component architecture for high-performance scientific computing". In Conference on High Performance Distributed Computing, 1999 [10] The Common Component Architecture Technical Specification – Version 0.5. http://cca-forum.org/bindings/old-0.5/.
[10] M. Vanneschi "Grid.it : a National Italian Project on Enabling Platforms for High-performance Computational Grids" GGF International Grid Summer School on Grid Computing Vico Equense Italy July 2003
www.dma.unina.it/~murli/SummerSchool/session-14.htm.
[11] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, C. Zoccolo "Grid.it component model "Project Grid.it WP8 Deliverable, Jan 2004.
[12] A. Machì, F. Collura "Skeleton di componenti paralleli riconfigurabili su griglia computazionale map e farm ".TR ICAR-PA-12-03 - Dec 2003.
[13] M. Cannataro, C. Comito, A. Congiusta, G. Folino, C. Mastroianni, A. Pugliese, G. Spezzano, D. Talia, P.Veltri "Grid-based PSE Toolkits for Multidisciplinary Applications" Working Paper GRID.it WP8 Dec 2003
[14] The Adaptive Communication Environment http://www.cs.wustl.edu/~schmidt/ACE.html
[15] E. Gamma, R. Helm, R. Joyhnson, J. Vlissides **"**Design Patterns . Elements of Reusable Object-Oriented Software". Addison-Wesley.
[16] Douglas C. Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann "Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects" Wiley & Sons in 2000, ISBN 0-471-60695-2.