A Distributed and Heuristic Policy-based Management Architecture for Large-Scale Grids

Edgar Magaña Perdomo

Thesis Director – Dr. Joan Serrat Fernández

A thesis submitted for the degree of
Doctor per la Universitat Politècnica de Catalunya

February 2008

Acknowledgements

The development of this research work, despite being presented by only one author, is the result of many contributions from countless extraordinary comrades whom I would like to thank for their priceless friendship and deep love.

I would like to thank my parents for their infinite confidence in my capacity to reach all my personal goals. I am sure that without your support this work would never have been done. Ana, thank you for being a lovely mother. I also thank you for your advice and mainly for all the sacrifices that you have made for me. Paco, you knew that I could do it. Thank you for keeping faith in me. Tita, you are the best sister and indefatigable hard working mother. Please, never forget I love you so much all of you.

A great woman has been to my side along the hardest moments of this work. Clara, I thank you for listening and advising me, you know very well that a huge part of this work is just because of you. I love you dear.

I would like to thank my advisor and thesis director Joan Serrat for his continuous support.. I thank Joan for our long discussions that helped improve this research and for his moral support when our results were not as we expected.

Research work in the telecommunications area is highly improved when it receives contributions from researchers with strong experience in new information technologies projects. I would like to thank to Masum Hasan for his contributions. I also would like to thank to Cisco Systems, Inc. for offering me the opportunity to move this research to a real commercial environment.

The most important results and contributions of this research are thanks to the support of my friend and colleague Laurent Lefevre. This work has been possible because of the collaboration with the Ecole Normale Supérieure de Lyon (ENSL).

The most memorable moments that I have experienced since I started the development of this thesis are thanks to my friends in Barcelona. I have enjoyed plenty of experiences with each one of you and there are not words to thank you for all the good and bad moments that we have had together. I really thank you, especially because you have stood by me no matter whether my mood was friendly or not. I really appreciate you, thanks for being always and unconditionally there.

Distance is the worst enemy when loneliness is living with you day to day. This is why I would like to thank my friends in Mexico who always remember me. I thank you for your emails and calls. I thank you for your advice and extraordinary friendship when I had the opportunity to reduce that cruel distance between us. Thank you for always offering me your time and love. Thank you very much, my friends.

This research work was partially supported by the Departament d'Universitats, Recerca i Societat de la Informació of the Generalitat of Catalonia, Spain.

It also received support from the National Council for Science and Technology (CONACYT) of Mexico (Register Number: 178575).

The work of this paper is partially supported by the Ministerio de Educación y Ciencia by means of the approved project TSI2005-06413. Moreover, this thesis was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

Some experiments of this article were performed on the Grid5000 platform, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (https://www.grid5000.fr).

Finally, this work also received support from the Public Education Secretary (SEP) of Mexico.

Agradecimientos

El desarrollo de este trabajo de investigación, a pesar de ser presentado por un solo autor, lleva consigo la contribución de un incontable número de personas a las que deseo agradecer su valiosa amistad y profundo cariño.

Quiero profundamente agradecer a mis padres por su infinita confianza en mi capacidad para concluir exitosamente este apasionante proyecto. Gracias a su apoyo logré seguir adelante a pesar de lo complicado que ha sido y lo muy desesperado que me llegue a encontrar. Ana, gracias por ser una madre con tanto amor y cariño. Gracias por los consejos y por todos los sacrificios que has hecho por mí. Paco, sabias que podía lograr doctorarme y te agradezco que siempre hayas mantenido esa fe. Tita, eres la mejor hermana y una mamá súper trabajadora. No olviden que los quiero mucho a los tres.

Una gran mujer ha estado a mi lado durante los momentos más duros de este trabajo. Clara, te agradezco que siempre me hayas escuchado y orientado, bien sabes que gran parte de este trabajo es gracias a ti. Te quiero mucho flaquita.

Me gustaría agradecer el constante apoyo de mi tutor y director de tesis Joan Serrat. Gracias Joan por apoyarme y ofrecerme tu orientación durante las largas discusiones para el perfeccionamiento de mi investigación y sobre todo el apoyo moral cuando los resultados no siempre eran los esperados.

Un trabajo científico en el área de las telecomunicaciones se ve altamente beneficiado cuando tiene el respaldo de expertos con experiencia en proyectos sobre nuevas tecnologías de la información. Me gustaría agradecer a Masum Hasan por las contribuciones que he recibí para mejorar mi trabajo. También quiero agradecer a Cisco Systems, Inc., por brindarme la oportunidad de extrapolar esta investigación en un ambiente orientado al mundo empresarial.

Los resultados más importantes de esta investigación son gracias al apoyo de mi amigo y colega Laurent Lefevre. Gracias a la colaboración con la Ecole Normale Supérieure de Lyon (ENSL) y el escenario de pruebas Grid5000 he podido presentar un trabajo funcional en un entorno real.

Los más memorables momentos que he vivido durante el desarrollo de esta tesis se los debo a todos mis amigos en Barcelona. A lo largo de todo este tiempo he disfrutado de tantas experiencias con cada uno de ustedes que no hay palabras para agradecer los buenos y malos momentos que hemos vivido. Les doy las gracias especialmente por humanamente soportar los momentos de mucho estrés y presión que he sufrido durante mi agradable estancia en esta bella cuidad.

La distancia es un enemigo cuando la soledad vive contigo día a día. Por eso quiero agradecer a mis amigos en México que siempre se acordaron de mí. Por sus correos y llamadas, por sus consejos y buena compañía cuando tenía la oportunidad de reducir esa cruel distancia que nos separó durante este tiempo, por siempre ofrecerme incondicionalmente su compañía y cariño. Muchas gracias compadres.

Este trabajo de investigación recibió apoyo del Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya, España.

Este trabajo ha sido posible gracias al apoyo recibido por el Consejo Nacional de Ciencia y Tecnología (CONACYT) de México. (Número de Registro: 178575).

El trabajo de investigación presentado en esta tesis ha estado parcialmente patrocinado por el Ministerio de Educación y Ciencia de España por medio del proyecto: TSI2005-06413. Además, esta tesis ha gozado del apoyo de la red de excelencia: EC IST-EMANICS (#26854).

Algunos de los experimentos presentados en esta tesis fueron realizados en el escenario de pruebas Grid5000. Un proyecto financiado por el ministerio de investigación de Francia por medio de la iniciativa ACI GRID, INRIA, CNRS y RENATER (https://www.grid5000.fr)

Finalmente, este trabajo ha recibido apoyo de la Secretaría de Educación Pública (SEP) de los Estados Unidos Mexicanos.

Abstract

Grid Computing is defined as a heterogeneous, distributed and shared system where applications are solved in dynamic, multi-institutional virtual organizations (VOs). This key concept involves the inherent ability to negotiate resource-sharing arrangements among a set of participating parties (providers and costumers) and then to use the resulting resource pool for some purpose. Basically, the Grid should integrate and coordinate resources and users that live within different control domains. Besides, it is built from multi-purpose protocols and interfaces that address such fundamental issues as scheduling, security, resource discovery, and resource allocation. Finally, the Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

Large-scale Grids are formed by thousands of nodes sharing multiples kind of resources (computing, networking, software applications, high-technology instruments, etc) and therefore, the total amount of shared resources become millions of individual entities that most be adequately integrated and coordinated for solving multi-disciplinarian problems. In this dynamic, heterogeneous and geographically dispersed environment, Resource Management (RM) is regarded as a vital component of the Grid Infrastructure.

Grid Resource Management (GRM) coordinates and shares multiple kinds of resources efficiently; and, in the above mentioned environments, GRM faces several challenges that make the implementation of practical systems a very difficult problem. Among these challenges we would like to emphasize the fact that GRM systems must fulfil strict functional requirements from heterogeneous, and sometimes conflicting, domains (e.g., the users', applications and networks domains). In addition, GRM systems must adhere to non-functional requirements that are also rigid, such as reliability and efficiency in terms of time consumption and load on the host nodes.

The aim of this thesis is to design and implement a new Grid Resource Management methodology, where non-massive resources owners would be able to share their resources and integrate human collaboration across multiple domains regardless of network technology, operative platform or administrative domain.

This thesis proposes a distributed and heuristic policy-based resource management architecture for large-scale Grids. The resource management architecture proposed herein is composed of four main building blocs: services management, resource discovery and monitoring, resource scheduling and jobs allocation and activation. The Grid Services Management (GSM) and Jobs Allocation and Activation (JAA) are supported by means of a Policy-based Grid Resource Management Architecture (PbGRMA). This architecture is able to identify service needs arising from diverse sources during the deployment and management of Grid Services, such as requirements demanded by customers, applications and network conditions. Afterwards, the PbGRMA merges these requirements into deployment policies for the corresponding Grid Services. The Grid Resource Discovery and Monitoring (GRDM) is supported by the introduction of the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS), in which network and computational resources are monitored by distributed agents. This allows for a flexible, heterogeneous and scalable monitoring system. The Grid Resource Scheduling (GRS) is based on the Balanced Load Multi-Constrained Resource Scheduler (BLOMERS). This heuristic scheduler represents an alternate way of solving the inherent NP-hard problem for resource scheduling in large-scale distributed networks by means of the implementation of a Genetic Algorithm.

Finally, based on the outcome of both the GRDM and GRS, the PbGRMA allocates the corresponding Grid Services by means of its interfaces with Globus ToolKit Middleware and Unix-based CLI commands along of any large-scale Grid Infrastructure.

The synergy obtained by these components allows Grid administrators to exploit the available resources with predetermined levels of Quality of Service (QoS), reducing computational costs and makespan in resource scheduling while ensuring that the resource load is balanced throughout the Grid. The makespan of a schedule is the time required for all jobs to be processed when no one job could be interrupted during its execution and each node can perform at most one operation at any time.

This new approach has been successfully tested in a real large-scale scenario such as Grid5000[1]. The results presented along this Thesis show that our general solution is a reliable, flexible and scalable architecture to deploy and manage Grid Services in large-scale Grid Infrastructures. Moreover, the substitution of the heuristic algorithm approach used into the Grid Resource Scheduling (GRS) phase by other non-heuristics selection algorithms could make our solution useful in smaller Grid Infrastructures.

[1]The Grid5000 is an initiative of the French Ministry of Research, and is supported by the ACI GRID action, INRIA, CNRS and RENATER.

Table of Contents

List of Figures

List of Tables

List of Acronyms

BLOMERS	Balanced Load Multi-Constrained Resource Scheduler
CISCO IP SLA	CISCO IOS® IP Service Level Agreements
CORBA	Common Object Request Broker Architecture
DiffServ	Differentiated Services
DSCP	DiffServ Code Point
EGA	Enterprise Grid Alliance
FAIN	Future Active IP Networks
GGF	Global Grid Forum
GI	Grid Infrastructure
GIP	Grid Infrastructure Providers
GIS	Grid Information Services
GRDM	Grid Resource Discovery and Monitoring
GRM	Grid Resource Management
GRS	Grid Resource Scheduling
GSC	Grid Services Customers
GSM	Grid Services Management
GUI	Graphical User Interface
HPC	High Performance Computing
HTML	Hyper Text Markup Language
IT	Information Technologies
JAA	Jobs Allocation and Activation
LDAP	Lightweight Directory Access Protocol
MIB	Management Information Base
MDS	Meta-Directory System
MonWG	Monitoring Working Group
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture
P2P	Peer-to-Peer
PbGRMA	Policy-based Grid Resource Management Architecture
PbM	Policy-based Management

QoS	Quality of Service
RDDTool	Round Robin Database Tool
RM	Resource Management
RMI	Remote Method Invocation
RTTMON-MIB	CISCO Round-Trip Monitoring MIB
SBLOMARS	SNMP-based Balanced Load Monitoring Agents for Resource Scheduling
SETI	Search for Extra-Terrestrial Intelligence
SLA	Service Level Agreement
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architectures
SOAP	Simple Object Access Protocol
TMN	Telecommunications Management Network
VO	Virtual Organization
WS-RF	Web Services Resource Framework
WSDL	Web Services Definition Language
WSS	Web Services Security
XML	eXtensible Markup Language

Chapter 1

INTRODUCTION

Today, computing systems are involved in nearly all human activities. This evolution on computational systems started when typical typewriters were replaced by personal workstations, home phones landlines were replaced by mobile phones with versatile features and sophisticated digital devices are used rather than simple electronic appliances. With so much change, the need to enable communication mechanisms between technological resources was inevitable. The communication between them improves their specific and individual aims in terms of time, cost and effectiveness. As a result of the inevitable necessity of communication mechanisms between computational systems, networks of computers were implemented.

Networks of computers are everywhere: mobile phone networks, corporate networks, factory networks, campus networks, home networks, in-car networks, on board networks in airplanes and trains. Unfortunately, computing networks started speaking their own languages and communication between them became difficult. Therefore, the possibility to exploit every network's advantages at the highest level was limited. This was inefficient, because computing systems could not perform more work that the one directly assigned.

When networks of computers were created to collaborate between each other, there was a clear improvement in the performance of every computational system forming these networks. This improvement was due to the fact that complex tasks were distributed along different parties of the network and then each of them executed a small number of these tasks, achieving faster results without overloading one system of the entire network. It was then that computing systems started to exchange dynamic information regarding activities and processes that they had to perform. Then they could coordinate these activities between each other and facilitate the work because it is shared between all the parties in the network. This concept marked the beginning of distributed systems.

Unfortunately, the simple fact of implementing a distributed system is not enough to guarantee an improvement of the resources consumption of the network. It is also necessary to implement efficient communication mechanisms and coordination entities in charge of distributed computing applications trough all the entities forming the network. A good example of this inherent problem can be found in any network of computers. For instance, workstations in a typical services enterprise are doing nothing 90% of the time [Scott_93]. The vast majority of the machines are used mostly for word processing, web browsing, downloading, etc. Not exactly stuff which grinds the processors or anything. At night, none of the computers are used at all. This means that there is a huge amount of processor power going to waste in these networks.

This waste can be seen around the world. There is a huge amount of un-harnessed power to be taken advantage of. At the same time, there are many tough problems which ordinary computers cannot solve. In order to improve the communications mechanisms between networks of computers, the formal concept of the distributed system has been introduced. A distributed system is a collection of independent computers that appear to the users of the network as a single computer in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing [Tan01].

Distributed systems like SETI@home [SETI] or RC5 [RC5]split up difficult tasks and tough problems to run on home PCs during screensaver time or when the processor is idle, and then collate all the information they receive back. These systems are traditionally based on the premise that many heads are better than one.

Grid Computing is an extraordinary example of the distributed systems concept. A Grid uses the resources of many separate computers connected by a network (usually the Internet) to solve large-scale computation problems. Most Grids use idle time on many thousands of computers throughout the world. Such arrangements permit handling of data that would otherwise require the power of expensive supercomputers or would have been impossible to analyze. Grid Computing is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements [Buy02].

1.1 GRID COMPUTING

The Grid Computing [Fos01] concept was introduced to denote a proposed form of distributed computing that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations. Grid technologies promise to change the way organizations solve complex computational problems. Nevertheless, it is an evolving area of computing, where standards and technology are still being developed to enable this new paradigm. The impact of Grid Computing in the academic and industrial sectors is growing every day, although it was originally understood as an emerging technology that provides an abstraction for resource sharing and collaboration across multiple administrative domains. Currently, the development of standards such as the Open Grid Service Architecture (OGSA) [Fos02a] along with the introduction of new paradigms such as the Semantic Grid [Bac04] is leading the Grid towards an environment that is not only suited for computational-intensive applications, but also for computing scenarios typical of distributed systems. These standards and emerging paradigms include service and information providing, multimedia environments and pervasive computing. For these and other reasons, Grid Computing is becoming an interesting and challenging environment supporting both old and new services for cooperative applications.

The essential mode to implement Grids is through Virtual Organizations (VOs) [Fos03]. These are groups of individuals, institutions and organizations collaborating to share the resources that they own in a controlled, secure and flexible way, usually for a limited period of time. This sharing of resources involves direct access to computers, software, and data. The coordinating and sharing of resources inherent in Grid networks introduces challenging resource management problems due to the fact that many applications need to meet stringent end-to-end performance requirements across multiple network instances. Furthermore, they should also

guarantee fault tolerance and network level Quality-of-Service (QoS) [ITU00] parameters as well as arbitrate conflicting demands.

Therefore, resource management systems are an important entity in charge of managing the shared resources and controlling various applications running in a distribute way along the members of a VO. These management frameworks have very heterogeneous environments to handle, they should be compatible with the emerging Grid Services definitions [Cza06]] and they have to be aware of the innovations in Web Services technology [Ved02], in order to fulfil the implicit requirements by New Generation of Grids (NGG) [NGG06].

1.2 RESOURCE MANAGEMENT IN LARGE-SCALE GRIDS

The concept of resource management makes use of a set of terms that are convenient to keep in mind in order to clarify the reading of this thesis. These terms are defined below:

- **Job:** A user-defined task that is scheduled to be carried out by an execution subsystem. In OGSA, a job is modelled as a manageable resource, has an endpoint reference, and is managed by a job manager. Some examples are executing certain application, storage users data and forward data packages through the network.
- **Resource:** It is something that is required to carry out an operation. Some examples are disk space, CPU cycles per second, available memory, and end-to-end performance metrics.
- **Computational Node (node):** This is an entity where different resources are physically allocated. The node is the identification point for any resources, so a node should have at least one resource shared to being considered part of the Grid Infrastructure.
- **Grid Infrastructure:** It is the whole set of resources shared for building a specific Grid.
- **Web Services:** A software system designed to support interoperable Machine-to-Machine interaction over a network. The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP standard and written based on the Web Services Description Language (WSDL) [Ved02].
- **Grid Services:** In general use, a Grid Service is a Web service that is designed to operate in a Grid environment, and meets the requirements of the Grid(s) in which it participates [Fos02a].
- **Scheduling:** The mapping of jobs to resources.
- **Workflow:** This is the ordering of a set of jobs for a specific user.
- **Grid Scheduling:** It is the mapping of individual Grid Services to network and computer resources, while respecting Service Level Agreements (SLAs).
- **Scheduling Constraints:** scheduling constraints can be hard or soft. Hard constraints are rigidly enforced. Soft constraints are those that are desirable but not absolutely essential. Soft constraints are usually combined into an objective function (Policies).

- **Scheduling Targets:** The goals to be fulfilled after the scheduling process. They may consist of the minimisation of maximum latency, minimisation of the cost to users, maximisation of the profit, maximisation of personal or general utility, maximisation of resource utilisation, fairness, minimisation of variance, maximisation of robustness and predictability, minimisation of broken SLAs, etc.
- **Scheduling Data:** A variety of data that are necessary for a scheduler to describe the jobs and the resources: job length, resource requirements estimates, time profiles, uncertain estimate, etc.

Once familiarized with the terminology that we will use in the thesis, it is also important to understand what the general concept of Resource Management in Large-scale Grids is about. The term "resources" covers a wide range of concepts including physical entities like computation, communication systems, storage devices (databases, archives and instruments), individuals (people and their expertise), capabilities (software packages, brokering and scheduling services) and frameworks for access and control [Nab04]. Based on these requirements for Grids, it is clear that a key activity to improve the functionality of various applications running in dispread will be to design fast, efficient, scalable and reliable resource management architectures.

Grid Resource Management (GRM) is the central component of a Grid system. It involves managing of computational resources across multiple administrative domains. The main activities are to accept requests from users, match users' requests to available resources to which the users have access and schedule the matched resources.

Currently, Grids are sets of computational resources working together to reach parallel targets based on the principle of using their resources so long as they are available. The most complex activity for GRM is to schedule numerous computing, network and storage resources in order to reduce procurement, deployment, maintenance, and operational costs.

As defined in [Nab04] the term "Resource Management" refers to the operations used to control how capabilities provided by Grid resources and services are made available to other entities, whether users, applications or services. Resource management is concerned with the manner in which these functions are performed, such as when request operations start or how long it takes to complete.

In summary, Grid Resource Management is defined as the process of identifying requirements from several sources, monitoring resources in real time, matching resources to services and users' requirements, allocating applications to available resources following some quality of service policies, and scheduling resources over time in order to run customers' applications as efficiently as possible.

1.3 MOTIVATION OF THE THESIS

Grid Resource Management (GRM) is regarded as a vital component of the Grid Infrastructure. It faces several challenges that make the implementation of practical systems a problem. Among these challenges we would like to emphasize the fact that GRM systems must fulfil strict functional requirements from heterogeneous and sometimes conflicting domains (e.g., the users', applications and networks domains). In addition, GRM systems must adhere to non-functional requirements that are also rigid, such as reliability and efficiency in terms of time consumption and load on the host nodes.

GRM is a very challenging area where plenty of research projects are working on better solutions to this matter. Grid Resource Management is the most active part of Grid Computing, which also deals with security, data transfer and services virtualization [Jan07]. In Chapter 2, we will describe some of the most active and important Grid Resource Management approaches.

Despite the work done, none of the proposed solutions has proven to be good in all facets that ideally a GRM system should exhibit. The following are the non functional properties that can be used to rate a GRM solution:

- **Scalability:** Scalability means the ability of the GRM architecture to either handle growing amounts of resources (network, computational, human, etc.) in a graceful manner, or to be readily enlarged. The scalability of the architecture is essential in large-scale Grids because these Grids are formed by thousand of individual entities.
- **Flexibility:** Flexibility is the ability offered by Grid Resource Management systems to adapt (self-adapt) to resource state, new resources, incoming standards and changing technology.
- **Heterogeneity:** Heterogeneity is the ability to handle any type of resource regardless of its supporting platform, designer, provider or administrative domain owner. In Grid Computing heterogeneity is also an important issue, but some research projects do not take it so much into account based on the fact than their resources are homogenous or at least the have some standards for integrating new resources.
- **Efficiency:** Efficiency is the capability of a GRM system or architecture to keep at reasonable levels the time consumed to manage the Grid. Efficiency is also related to keep balanced the load offered to the managed resources.

Scalable systems have been addressed through resource management methodologies based on decentralized or hybrid schemes. Moreover, future solutions should be able to handle

incoming new devices and resources at the Grid. More specifically we can mention approaches based on distributed mechanisms such as P2P technologies [Ram05] or decentralized management agents [Nad06].

In terms of flexibility and efficiency, current GRM systems research is focused mostly on solving either one or the other of the two main problems at hand. The first of these is concerned with evenly balancing resource loads throughout a Grid. In this area, plenty of researchers have proposed a wide range of technologies with varying degrees of success. The second problem, however, involves shortening the makespan. The makespan of a schedule is the time required for all jobs to be processed by a set of unknown resources. For quite some time, a comprehensive solution addressing both of these problems proved elusive. In other words, a load-balanced solution with a minimal makespan was necessary.

Heterogeneity is a challenging issue in Grid Computing. We could affirm that most of the current solutions do not solve it because they are based on the assumption of a priori knowledge of device types and resources that the GRM should handle to facilitate network administrators tasks. Obviously, this solution is not possible in real large-scale Grids. New methodologies should be able to handle heterogeneous resources following the most common standards such as SNMP [Sta99], XML [Sch03] or Web Services [Ved02].

It is worthwhile to mention that although some Grid Resource Management projects are apparently excellent alternatives to be taken as de-facto standards, They have have important problems and failures. In fact, they may overload workstations or servers where they are being executed. Others are not heterogeneous in terms of the kind of resources they can handle or the processing time needed to perform the resource management process of the received services is quite long when hard constraints[1] or soft constraints[2] have to be considered. Both types of constraints are demanded by customers through requesting Grid Services.

Grid Services are Web Services [Ved02] with improved characteristics and features. Web services enable interoperability via a set of open standards, which distinguish them from previous network services such as Java's Remote Method Invocation (RMI) [Gro01], the Common Object Request Broker Architecture's (CORBA), or the Internet Inter-ORB Protocol (IIOP) [Orf00]. Web Services are platform independent and are commonly used for building emerging Service Oriented Architectures (SOAs) [Tho05].

Grid services can be "stateful" or "stateless". An instance of a service is stateless if it cannot remember prior events and an instance of service is stateful if it can remember prior

[1] Hard constraints are specific applications and resource requirements that must be fulfilled

[2] Soft constraints are management policies for resource utilization, deadlines and response time

actions (this implies variables within service that can maintain values between accesses). They are also “transient” or “non-transient”. A transient service instance is one that can be created and destroyed (usually, they are created for specific clients, which they do not outlive). An instance of a service is non-transient (persistent) if it outlives its client. Web Services are non-transient. They don’t support service creation and destruction.

In summary, resource management in large-scale Grids is a very challenging area where new methodologies are needed to improve current approaches in order to offer better solutions to small, medium and big companies who are not exploiting their resources at optimal level. With Grid Computing as a network technology, these companies will increase their efficiency and they will reduce their response time in all processes that are part of their business.

1.4 CONTRIBUTIONS OF THE THESIS

We propose an alternative solution to the state of the art in terms of Grid Resource Management, by means of splitting and distributing the resource management process in four main phases. We have introduced novelty contributions in each one of these phases in order to improve current systems or methodologies and cover the required features (efficiency, scalability, heterogeneity and flexibility) for Grid Resource Management in large-scale Grids.

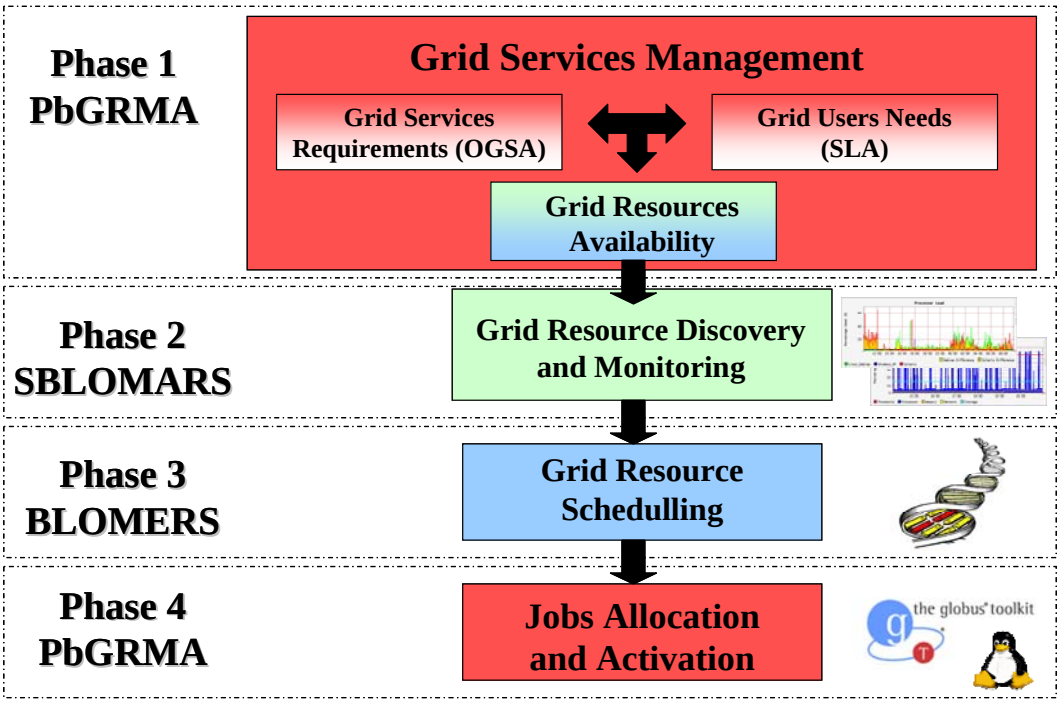

Figure 1.1 Grid Resource Management phases proposed in this thesis

The first phase is Grid Services Management (GSM) where both hard and soft constraints should be identified as requirements of certain Grid Services. The second one is Grid Resource Discovery and Monitoring (GRDM) where resources offering their individual capacities to the Grid under certain conditions are discovered and monitored in real time. The third one is to schedule received applications into the set of available resources carrying out both constraints the hard ones and soft ones, it is known as Grid Resource Scheduling (GRS). The fourth and last phase is Grid Jobs Allocation and Activation (JAA), which is in charge of allocating received jobs into the selected set of available resources including files staging and cleanup. In the schema on Figure 1.1 we show the orchestration of these phases.

These phases are not sequential at all. The GRDM is running all time and reporting resource availability information through its graphical interface. On the contrary, the rest of the phases are waiting to receive inputs from higher level entities (Figure 1.1).

Grid Services Management (GSM) and Jobs Allocation and Activation (JAA) phases are solved by a Policy-based Grid Resource Management Architecture (PbGRMA) [Maga07a]. On one hand, this architecture identifies Grid Services requirements from three different sources: the users' QoS needs, resource provider's availability (i.e. amount of resources free to execute new services) and service specifications according to Open Grid Services Architecture (OGSA). On the other hand, the PbGRMA merges these requirements creating a Grid Domain Level policy for the corresponding Grid Service. It is the starting point of the Grid Resource Management process. It is also worthy to mention that we have made this PbGRMA compatible with Globus Middleware [Bor05] and Linux CLI [Wee04] to allocate and execute Grid Services along with large-scale Grid Infrastructures.

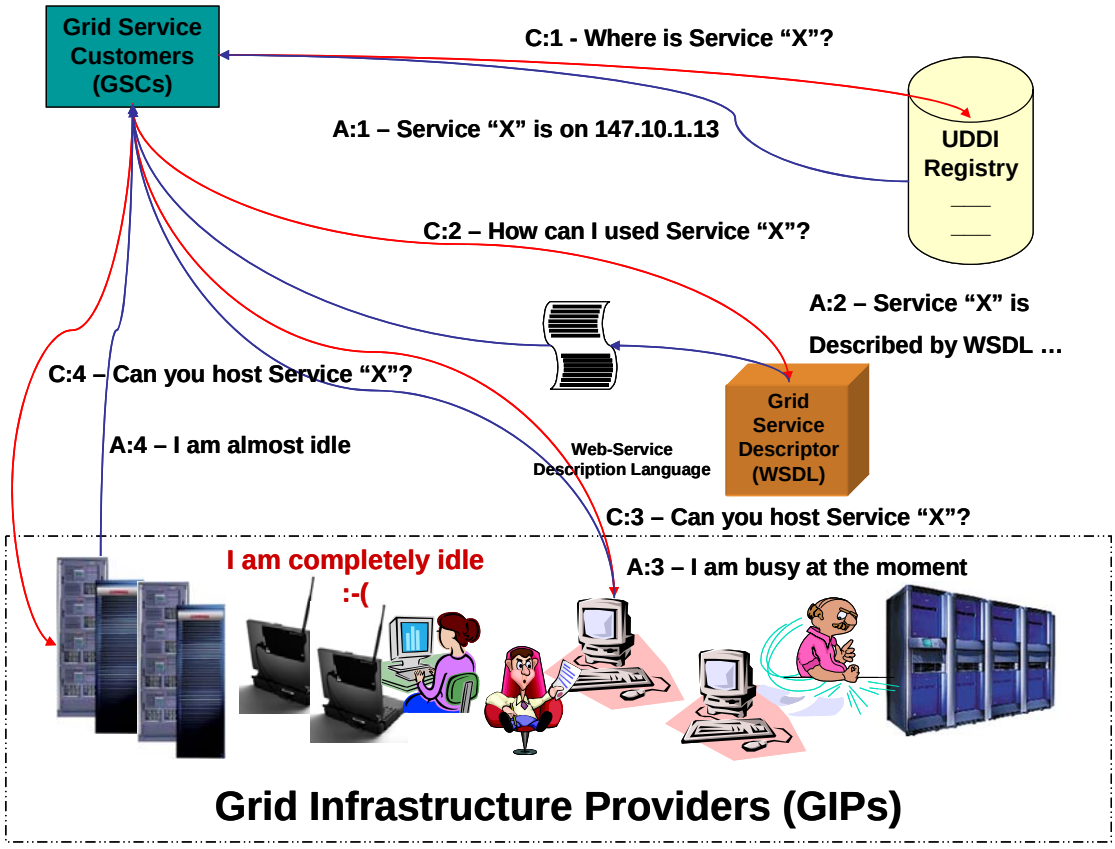

Figure 1.2 Current business model for Grid Services Management

PbGRMA has been successfully used in other environments such as Active Networks [Sal03]. This architecture is characterized by a reliable and autonomous deployment, activation and management of Grid Services. Although applicable to any user profiles, our system is essentially intended for non-massive resource owners accessing large amounts of computing, software, memory and storage resources. Unlike similar architectures, it is able to manage service requirements demanded by users, providers and services themselves. This architecture is also able to manage computational resources in order to fulfil QoS requirements based on a balanced scheduling of resources exploitation. It is flexible because it is able to self-extend by incorporating management components and policies interpreters needed to control multiple infrastructures regardless operative platform or network technology.

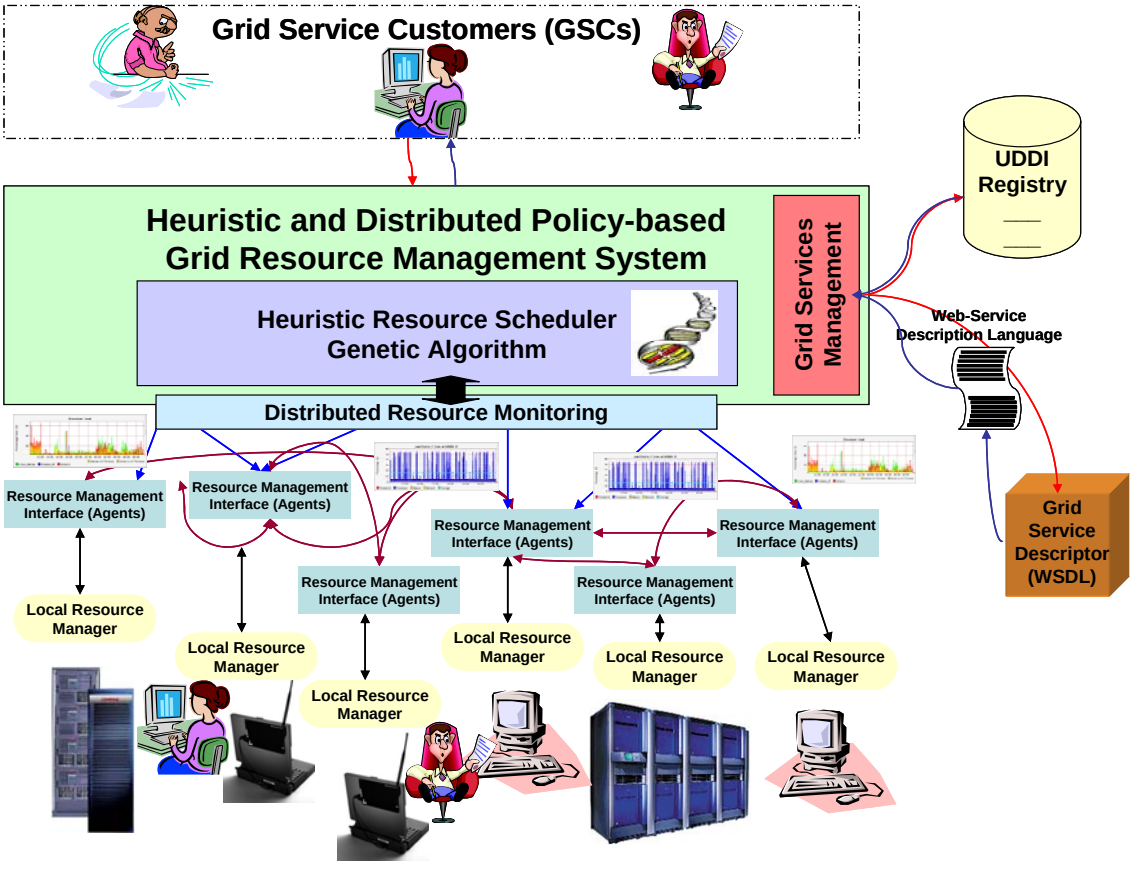

Figure 1.3 Business model proposed for Grid Services Management

One of the most important contributions in our Policy-based Grid Resource Management System design is that simplifies the communication between Grid Services Customers (GSCs), Grid Services Repositories and Grid Infrastructure Providers (GIPs). The current process for

deploying Grid Services on Grid Infrastructures involves a set of sequential steps which require the interaction of the clients many times. Normally, the GSC has to first invoke a service name registry, then request a service instantiation and finally he receives the Grid Service handler. These interactions are shown in Figure 1.2.

The solution proposed into the PbGRMA simplifies this process from the GSC's and GIP's point of view. It is designed to retrieve the Grid Service handler from the UDDI Registry and the Grid Service requirements from the Grid Service Descriptor Server on behalf of the GSCs. In Figure 1.3 we are illustrating the business model evolution that we are proposing in our solution in order to simplify Grid Services deployment in large-scale Grids.

We face the Grid Resource Discovery and Monitoring (GRDM) phase by introducing our SBLOMARS architecture that stands for SNMP-based Balanced Load Monitoring Agents for Resource Scheduling [Maga07b]. SBLOMARS monitoring system is a set of autonomous, distributed SNMP-based monitoring agents, which are in charge of generating real-time and statistical resource availability information for every resource and entity forming the Grid. So far, SBLOMARS is able to monitor processor, memory, network (interface level), memory, storage, applications and network (end-to-end networking traffic) for different architectures such Unix-based systems, Solaris, Microsoft-based and Macintosh systems. It is also self-extensible to multi-processor platforms as well as storage cluster systems. Its design is based on SNMP technology to achieve higher levels of heterogeneity and also based on autonomous distributed agents for scalability purposes in large-scale Grids.

SBLOMARS monitoring agents [Maga07b] consist of a set of distribute resource monitoring agents which are constantly capturing end-to-end network and computational resources performance. SBLOMARS faces the scalability problem by the distribution of the monitoring system into a set of sub-monitoring instances which are specific to each kind of computational resource to be monitored. Current monitoring systems GridRM [Bak05], Grid Monitoring [Bal03] and ReMos [Dew04] fail when new components are integrated into the Grid Infrastructure and some others fail when heterogeneous resources are operating into the Grid, thus laking of heterogeneity. Our approach reaches a high level of heterogeneity by means of the integration of the Simple Network Management Protocol (SNMP) that is commonly supported in almost any platform. SBLOMARS has been designed as a full independent and autonomic system. It could be re-configured by itself based on the performance load in its hosting nodes. SBLOMARS also introduces the concept of dynamic software structures, which are used to monitor everything from simple personal computers to complex multiprocessor systems or clusters with multiple hard disk partitions. These features make our approach novel compared with similar monitoring systems such as Ganglia [Mas04] or MonAlisa [Leg04]. In contrast to current monitoring

approaches, SBLOMARS integrates an end-to-end network-level monitoring technology; namely the CISCO IOS® IP Service Level Agreements [Cis06] that allows users to monitor end-to-end network-level performance between routers or from any remote IP network device.

The Grid Resource Scheduling (GRS) phase searches and matches jobs' requirements with resource availability. In other words, it determines which resources are best for executing a specific job, application or service. Our approach covers this phase by introducing the Balanced Load Multi-Constrain Resource Scheduler (BLOMERS) [Maga07c]. This scheduler makes use of the real-time and statistical resource availability information generated by SBLOMARS monitoring agents. BLOMERS implements a heuristic approach in order to tackle with the scalability problem and makes use of statistical resource availability information to do scheduling with load-balancing constraints.

The BLOMERS scheduler [Maga07c] deals with several conditions. It selects a set of candidates' resources from a poll, keeping individual resource performance comparatively balanced in all nodes of the Grid. This condition has been added in order to satisfy the computational resource load balancing. In BLOMERS approach, we propose to find a sub-optimal solution to the problem of scheduling computational resources. BLOMERS scheduling system is based on a Genetic Algorithm (GA) [Pag05] in charge of resource selection, as a part of the resource manager system. So far, BLOMERS is already implemented and running in both a local small-scale test-bed and a real large-scale test-bed. It has been compared from quantitative and qualitative points of view in the following paper [Maga07d]. We have also executed scheduling experiments in the Grid5000 test-bed [Cap05] as a complementary activity for this approach.

BLOMERS as a heuristic-based resource scheduler has important contributions regarding makespan saving and balancing the resource load along the Grid. These contributions are possible because of the integration and combination of two methodologies for generating the new set of candidates (recombination operators). In heuristic models, the number of solutions to check, in order to see if they are matching resources requirements, is reduced by means of selecting smaller patterns from the full set of resources available in the Grid. Normally, heuristics solutions fail to apply their recombination operators because they do not take into account the statistical usability of the resources. In our approach we are using statistical information from SBLOMARS monitoring agents in order to improve our new possible resources.

1.5 STRUCTURE OF THE THESIS

This thesis is composed by seven Chapters and three Appendixes as follows:

Chapter 1 introduces the Thesis, providing a general outline of Grid Computing technology and the resource management problem in large-scale Grids. We also state the motivations of the Thesis and its main contributions to solve the inherent problems in resource management.

Chapter 2 contains a review of the literature related to Grid Resource Management. In this Chapter we will describe some of the most important and active projects which are working on improving resource management in Grid Computing.

Chapter 3 describes the first contribution of the Thesis, the Policy-based Grid Resource Management Architecture (PbGRMA). We will describe the mechanisms implemented in this architecture to cover two of the four proposed resource management phases, Grid Services Management (GSM) and Jobs Allocation and Activation (JAA).

Chapter 4 describes the second contribution of the Thesis, the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling in Large-scale Grids (SBLOMARS). These distributed monitoring agents are solving the second phase proposed in this Thesis, the Grid Resource Discovery and Monitoring (GRDM).

Chapter 5 describes the third and last contribution of the Thesis, the Balanced Load Multi-Constrain Resource Scheduler in Large-scale Grids (BLOMERS), a heuristic approach to solve the NP-hard problem for scheduling huge amount of network and computational resources in Grids.

Chapter 6 contains evaluations scenarios and examples for each one of the contributions of the Thesis. In this Chapter we will describe the experiments executed to show the advantages of our approaches and contributions.

The thesis ends with Chapter 7, where the main results of the thesis are summarised. In this Chapter we also express our conclusions and the future work that could be done in this area.

Appendix A is a full Grid Computing background in order to offer introduction information on the Grid Computing field for new researchers on this area.

Appendix B is a brief background of Genetic Algorithms technology. It is important to read this information to understand the terminology and fundamentals of this approach.

Appendix C is a list of the research publications by the author of the thesis.

Chapter 2

REVIEW OF THE LITERATURE

2.1 INTRODUCTION

Grid Computing in distributed systems world was formulated in the last decade. It referred to an envisioned advanced distributed computing paradigm with capabilities to ultimately assist in solving complex science and engineering problems beyond the scope of existing computing infrastructures [Fos03]. The concept has evolved considerably since that time. Grid computing is gaining a lot of attention within the IT industry and some others [Ram03]. Although it has been used within the academic and scientific community for some time, standards, enabling technologies, toolkits, and products are becoming available that allow businesses to use and reap the advantages of Grid Computing.

The growing popularity has also resulted in various kinds of ‘Grids’, commonly known as Data Grids, Distributed Computational Grids, High Performance Computing Grids, Cluster Grids and Enterprise Grids, among many others [Fos02b]. Efforts are in progress to converge the concepts related to the architecture, protocols, and applications of these grids to formulate a single paradigm: The Grid.

This chapter summarizes the evolution of the Grid Computing concept along the last years and proposes taxonomy of the types of Grids, which currently are the most commonly implemented. It is worth understanding this taxonomy because Resource Management features and requirements are different for each one of these Grids. Moreover, we will describe the main factors, which we have used to decide the type of Grid that we finally have worked on (Distributed Computational Grids). Nevertheless, the focus is on the management functions that should be performed by the Grid Resource Management (GRM) systems in Distributed Computational Grids. Once we have highlighted the most critical of these activities, we will explain what work has been done on them in order to perform the GRM process in large-scale computational Grids. In addition, we will go through the different lines proposed to generalize the resource management problem.

The purpose of this chapter is to describe our main motivation to improve resource management systems. These systems are intended to be in charge of executing most of the resource management operations on behalf or their users. They must provide support for some form of scheduling, monitoring, allocation and discovering computational and networking resources. They also should provide mechanisms for interfacing Grid Service Customers (GSCs) with Grid Infrastructure Providers (GIPs) as well as identifying and allocating of Grid Services, which have been demanded by GSCs

There are many architectures and different global aims in the Grid world. Beyond the fact that it would be ponderous to perform a review of the existing literature in all these architectures, it would not be possible to design a full and generic Grid Resource Management System covering all issues related to all of these types of Grids. During this research we discovered that Distributed Computational Grids are the least used and implemented grid architectures. Moreover, this type of Grid needs sophisticated mechanisms to perform management activities mainly because the amount of targeting resources is very large. Distributed Computational Grids are also called Large-scale Computational Grids. For all these considerations, we decided to focus our research on this type of grid.

Large-scale Computational Grids are composed of thousands of nodes that share multiple kinds of resources (e.g., computational and network resources, applications, instruments, etc.). Therefore, the total number of shared available resources comprises thousands of individual entities that must be adequately integrated and coordinated to enable the solution of multi-disciplinary problems.

This Chapter is composed of five sections. After Introduction, Section 2.2 describes our taxonomy proposal for current Grid Computing methodologies. Section 2.3 illustrates current Resource Management Architectures in Large-scale Distributed Computational Grids. In Section 2.4 we proceed to explain Resource Management Functions and the review of the literature on each one of them. Finally, Section 2.5 concludes this Chapter.

2.2 A TAXONOMY OF GRID COMPUTING

The article in [Fos02b] lists the three fundamental and basic characteristics that may be significant in determining whether a distributed computing system meets the requirements to be considered as a Grid. According to [Fos02b], such a system qualifies as a Grid, which:

I. Coordinates resources that are not under centralized control.
II. Utilizes standard, open, general-purpose protocols and interfaces.
III. Promises to deliver non-trivial qualities of service.

The most common description of Grid Computing includes an analogy to a power grid. When you plug an appliance or other object requiring electrical power into an outlet, you expect that there is power of the correct voltage available, but the actual source of that power is not known. Your local utility company provides the interface into a complex network of generators and power sources and provides you with (in most cases) an acceptable quality of service for your energy demands. Rather than each house or neighbourhood having to obtain and maintain its own generator of electricity, the power grid infrastructure provides a virtual generator. The generator is highly reliable and adapts to the power needs of the consumers based on their demand.

The vision of Grid Computing is similar. Once the proper kind of infrastructure is in place, a user will have access to a virtual computer that is reliable and adaptable to the user's needs. This virtual computer will consist of many diverse computing resources. But these individual resources will not be visible to the user, just as the consumer of electric power is unaware of how their electricity is being generated. To reach this vision, there must be standards for Grid Computing that will allow a secure and robust infrastructure to be built. Standards such as the Open Grid Services Architecture (OGSA) [Fos02a] and incoming technologies provide the necessary framework where businesses will build their own infrastructures. Over time, these infrastructures will become interconnected. This interconnection will be made possible by standards such as OGSA, Web Services and the analogy of Grid Computing to the power grid will become real.

According to this light summary of the evolution in the Grid Computing concept [Jac06], Grids are used in a variety of ways to address various kinds of application requirements. Often, Grids are categorized by the type of solutions that they best address. In our analysis we have identified four primary types of Grids which are summarized below. Of course, there are no hard boundaries between these Grid types and often Grids may be a combination of two or more of these.

2.2.1 High Performance Computing Grids

These Grids can be considered as the archetype of a Grid. Due to the complexity of many scientific problems, it is frequently impossible to obtain an analytical solution. Instead search heuristics are used to solve many optimization problems and simulation is often the method of choice to determine the behaviour of a complex process. These methods require more computing resources in order to solve more complex problems. In this type of grid, most of the machines are high-performance servers. The most publicly known Grid projects fall into this scenario in which different computing sites of scientific research labs collaborate for joint research. Here, compute-intensive and/or data-intensive applications are executed on the participating High Performance Computing (HPC) resources, which are usually large, parallel computers or cluster systems. The total number of participating sites in this Grid project is commonly in the range of tens or hundreds; the available number of processing nodes is in the range of thousands. Most people active in Grid research originate from this community and have these kinds of Grid scenarios in mind.

2.2.2 Distributed Computational Grids

Distributed Computational Grids are also called Scavenging Grids. This kind of Grid is most commonly supported by a large number of desktop machines. Machines are scavenged for available CPU cycles and other resources. Owners of the desktop machines are usually given control over when their resources are available to participate in the Grid. Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and high-performance orientation [Cab07].

Distributed Computational Grids aggregate substantial computational resources in order to solve problems that cannot be solved on a single computational system [Fos00]. These aggregated resources might comprise the majority of the supercomputers in the country or simply all of the workstations within a company. Here are some contemporary examples:

- Distributed interactive simulation (DIS) is a technique used for training and planning in the military. Realistic scenarios may involve hundreds of thousands of entities, each with potentially complex behaviour patterns. Yet even the largest current supercomputers can handle at most 20,000 entities. In recent work, researchers at the California Institute of Technology have shown how multiple supercomputers can be coupled to achieve record-breaking levels of performance.

- The accurate simulation of complex physical processes can require high spatial and temporal resolution in order to resolve large-scale details. Coupled supercomputers can be used in such situations to overcome resolution barriers and hence to obtain qualitatively new scientific results. Although high latencies can pose significant obstacles, coupled supercomputers have been used successfully in cosmology [Nor96], high-resolution chemistry computations [Nie96], and climate modelling [Mec99].

2.2.3 Data Grids

Data grids [Che05] are being built around the world as the next generation data handling systems for sharing, publishing, and preserving data residing on storage systems located in multiple administrative domains. A data grid provides logical namespaces for users, digital entities and storage resources to create persistent identifiers for controlling access, enabling discovery, and managing wide area latencies. A data grid provides virtualization mechanisms for resources, users, files, and metadata. Each virtualization mechanism implements a location and infrastructure-independent name space that provides persistent identifiers.

Data grids are perfect for organizations that need a collaborative work environment despite having diverse, distributed resources where data resides across multiple business and/or organizational domains. Data grid services allow users to access and manipulate data residing at sites around the world. Data can be retrieved from any location on the grid, and can be deposited or replicated to any location with space. A data grid is responsible for housing and providing access to data across multiple organizations. Users are not concerned with where this data is located as long as they have access to the data. For example, you may have two universities doing life-science research, each with unique data. A data grid would allow them to share their data, manage the data, and manage security issues such as who has access to what data.

2.2.4 Per to Per Systems (P2P)

Another common distributed computing model that is often associated with or confused with Grid Computing is Peer-to-Peer computing (P2P). In fact, some consider this another form of Grid Computing. A detailed analysis and comparison of grid computing and peer-to-peer computing is provided in [Iam03].

2.3 Resource Management Architectures in Large-scale Distributed Computational Grids

So far, we have explained the main differences between Data Grids, High Performance Grids and Large-scale Computational Grids. We have considered it extremely important to explain these differences before introducing the most fundamental Grid Resource Management (GRM) initiatives, which are based on the type of Grid being managed. As we have mentioned before, in our research we have focused on Resource Management for Large-scale Computational Grids. Therefore, we will highlight the main features and requirements of the Grid Resource Management System in Large-scale Computational Grids.

In Large-scale Computational Grids, the resource manager is one of the most critical components of the Grid, since it is responsible for handling applications requirements, selecting resources and scheduling jobs in complex environments [Min04]. Grid Resource Management (GRM) in Large-scale Distributed Computational Grids is the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring resources over time in order to run applications as efficiently as possible.

GRM process involves many details in its context. Basically, there are three involved actors in this definition. The first ones are Grid Service Customers (GSCs). They are the end users of the Grid. The second ones are Grid Infrastructure Providers (GIPs). They offer computational and networking resource such as processor, storage, bandwidth, memory and software. The last are Grid Services Repositories (GSRs). They are mainly databases where Grid Services are defined based on OGSA and WS-RF standards. They support the scalable evolution of Grid Services in heterogeneous domains and stand for the reliable functionality of Grid Services in real large-scale distributed computational Grids. These actors and their mutual interactions are shown in Figure 2.1. The Grid Resource Management mediates between GIPs (also called resource owners), GSCs (who are the customers) and GSRs (databases with Grid Services resource properties documents).

In managing the complexities present in Large-scale Computational Grids, traditional approaches are not suitable as they attempt to optimize system-wide measures of performance. Traditional approaches use centralized policies that need complete state information and a common resource management policy, or decentralized consensus-based policy. Due to the complexity in constructing successful Grid environments, it is impossible to define an acceptable system-wide performance matrix and common fabric management policy. Therefore, hierarchical and decentralized approaches are suitable for Grid resource and operational management [Buy99]. Within these approaches, there exist different models for management and regulation of supply and demand for resources [Buy103].

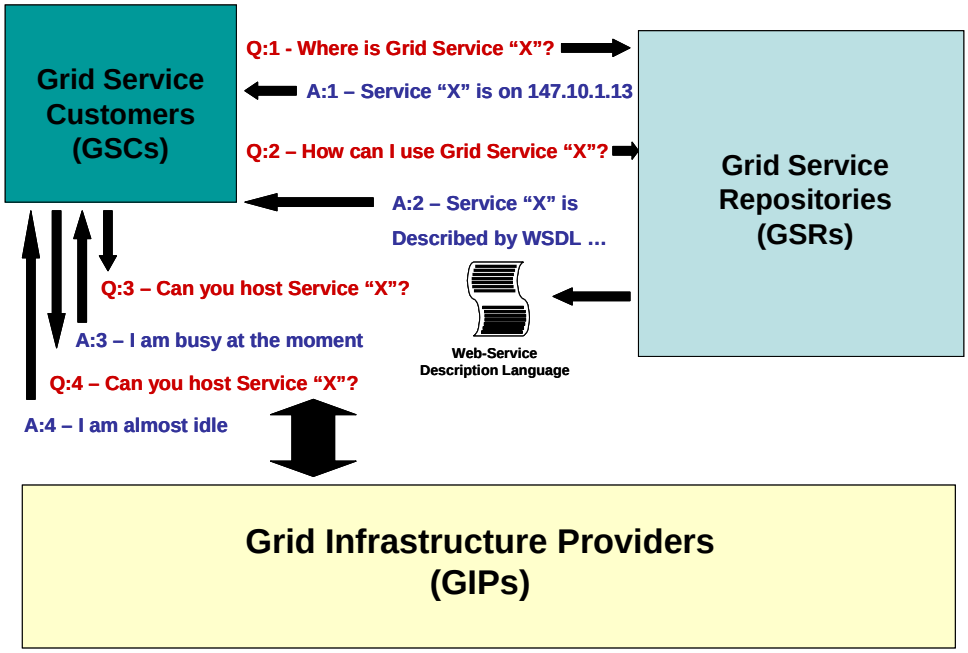

Figure 2.1 Large-scale Grid Resource Management actors

Large-scale Computational Grid environments contain heterogeneous resources, local management systems (single system image OS, queuing systems, etc.) and policies, and applications (scientific, engineering, and commercial) with varied requirements (CPU, I/O, memory, and/or network parameters). The Grid Infrastructure Providers (GIPs) and Grid Service Consumers (GSCs) have different goals, objectives, strategies, and demand patterns [Buy99]. More importantly, both resources and end-users are geographically distributed with different time zones. A number of approaches for resource management architectures have been proposed and the prominent ones are centralized, decentralized, hierarchical and mixed (better known as hybrid architectures).

2.3.1 Centralized Resource Management

A Central Resource Management System will be responsible for all the coordination and management issues, and will act as a middleware. It will receive the request from a client and broadcast to agents to ask which machine is best suitable for executing the application. After getting the responses from agents, the manager will apply the most suitable machine algorithm; will be developed in first stage of project to select a machine based upon the information received. Now the coordination between real client and the best machine will start to execute the application, either through the manager or direct coordination.

Unfortunately, this approach has fewer advantages. Its implementation is quite simple because only one centralized system is required and the control and vision of the resources is controlled from only one point. The performance of this approach is more reliable and fast when the requesting client is close to the central manager in terms of time, and when the connection between manager and client is rich in bandwidth.

This management system lacks fault tolerance, when the manager goes down, the overall scenario fails to perform. If many clients request to execute an application at same time, the server faces/manager faces the bottleneck in responses to the agents. Because if there are 'n' machines under a manager and any machine appears to ask for its application execution, a manager will generate 'n' possible requests to check who the best machine is and will have 'n' possible responses from agents. So imagine if 5 clients request a manager to find the best machine. Then it will generate 5*n requests and will receive 5*n responses from clients plus some extra time to apply rules and policies to select the best machine. This will put a mess of burden on the manager, and in our case we wish our manager to provide a service transparently to a client so that the client may not come to know that the particular application was executed on his machine or on a remote machine. Finally, this approach introduces more overhead on the network, as it requires more useless traffic roaming to check agents. If the central manager is far away, communication delay and communication bandwidth issues effects the performance.

2.3.2 Decentralized Resource Management

In a fully Decentralized Resource Management approach there are no dedicated resource controller machines, the resource requestors and resource providers directly determine the allocation and scheduling of resources. Logically there are as many different request and job queues as there are resource requestors and resource providers in the Grid.

In general, Decentralized Resource Management systems may lead to instability because of the lack of global perspective. The major drawback of a fully decentralized design is the increase in latency time and communication overhead to locate the requested object. Decentralized Resource Management systems integrate several subsystems and have a composite life cycle. The independent subsystems of these systems consist of uniform components, which are capable of interacting with each other to perform global functions. However, nothing monitors or controls these subsystems in an integrated manner. The monitoring system is only a subsystem and, like an organ in a living organism, there is no hierarchy of subsystems.

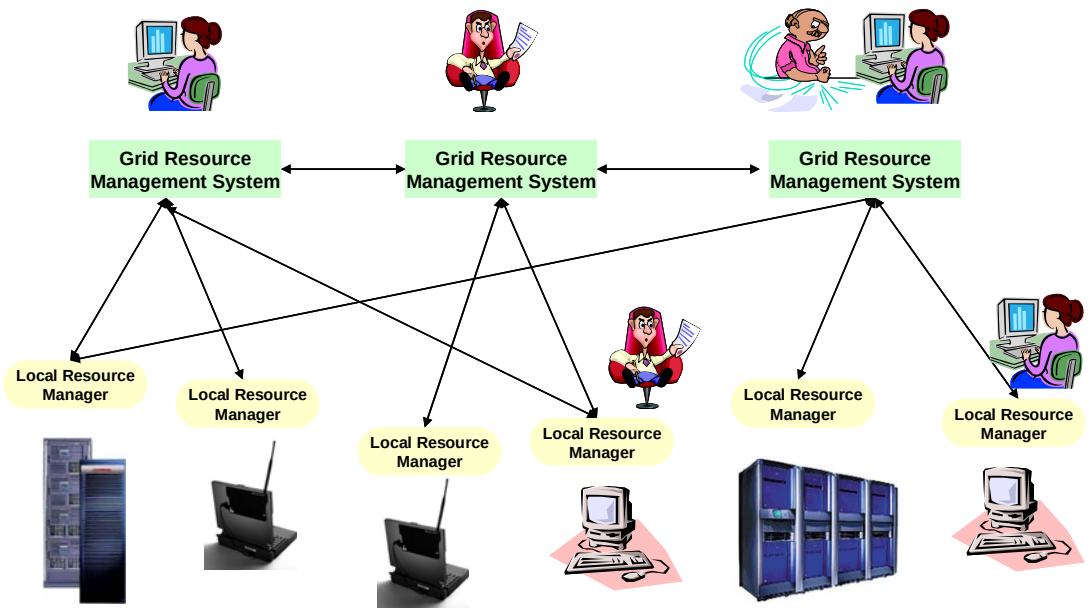

Figure 2.2 Decentralized Grid Resource Management system

2.3.3 Hierarchical Resource Management

In Hierarchical Resource Management Systems, the entities and components are organized in a hierarchy. High-level resource entities are managed at higher levels and lower-level, smaller sub-entities are managed at lower levels of the hierarchy. A Hierarchical Resource Management System is structured by a plurality of agents and sub-managers connected to lower communication networks and an integration manager connected to a higher communication network. Each of the sub-managers functions as an agent to the integration manager and functions as a manager to each agent.

On one hand, these systems avoid bottleneck at one server/manager (as compared to Central Resource Management systems), as it will not be the only machine in the overall network to entertain a client. Suppose if we have 5 managers and n clients, we call allocate responsibility to manager for machines depending on the rules and policies that we will design. For instance, a simple rule could be “Manager A” is responsible for n/5 machines. They provide quicker response to a client’s request to find a machine for application execution. The network overload is fewer busy as less useless traffic than real one, to check agents. The application’s execution is done in faster time due to the use of the nearest machine below the primary manager.

On the other hand, they need complex architectures from implementation prospective. When the primary manager fails to find a suitable machine, that is overhead from a performance point of view because the time will be considered unused as the higher manager will have to follow the same practice to find a machine for a client. Therefore, time spent by the lower manager will be wasted.

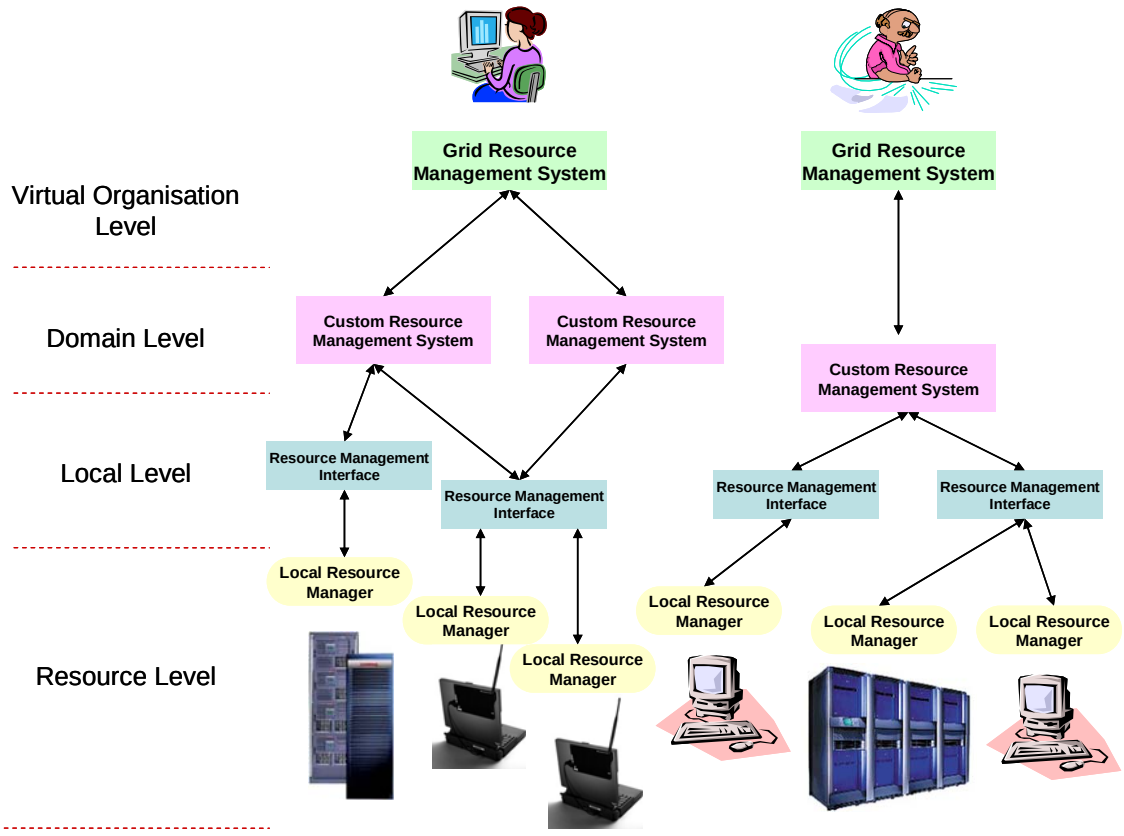

Figure 2.3 Hierarchical Grid Resource Management system

2.3.4 Hybrid Resource Management

Hybrid Resource Management is the newest alternative to the previous ones. In Hybrid Resource Management systems, local agents are controlling and monitoring shared resources and distributed resource managers are interacting with these agents in order to perform management operation like scheduling and evaluation. Basically, these systems integrate both solutions, the decentralized and the centralized, been the local agents the centralized system and the distributed resource managers the decentralized system. These systems could be very similar to decentralized systems but the main difference between decentralized systems and

hybrid systems is that, when a request is sent by a customer to find a best possible node to perform certain application or job, that request is received by every manager in the network and compared between all possible alternative in order to select the best one.

Hybrid management systems have the potential to dissolve perennial interface problems and create an environment that's customer focused. Real-time management operations are performed in hybrid resource management systems due to the fact that resources are on-fly monitored by decentralized agents. Therefore, hybrid resource management systems get real-time information about resources status. In these systems the bottleneck that central resource management systems have is avoided. Moreover, if one manager goes down, it does not affect the overall performance of the resource management system. Time wasted in Hierarchical Resource Management Systems is avoided when each individual manager starts to find out an available resource. Then, a faster service is offered to the customer. These systems adjust its Quality of Service (QoS) levels depending on the agreements between GSCs and GIPs. Finally, resource reservation requests are performed based on feasibility, optimality and stability analysis of the available resources. It also evaluates if a resource reservation request may be granted or not.

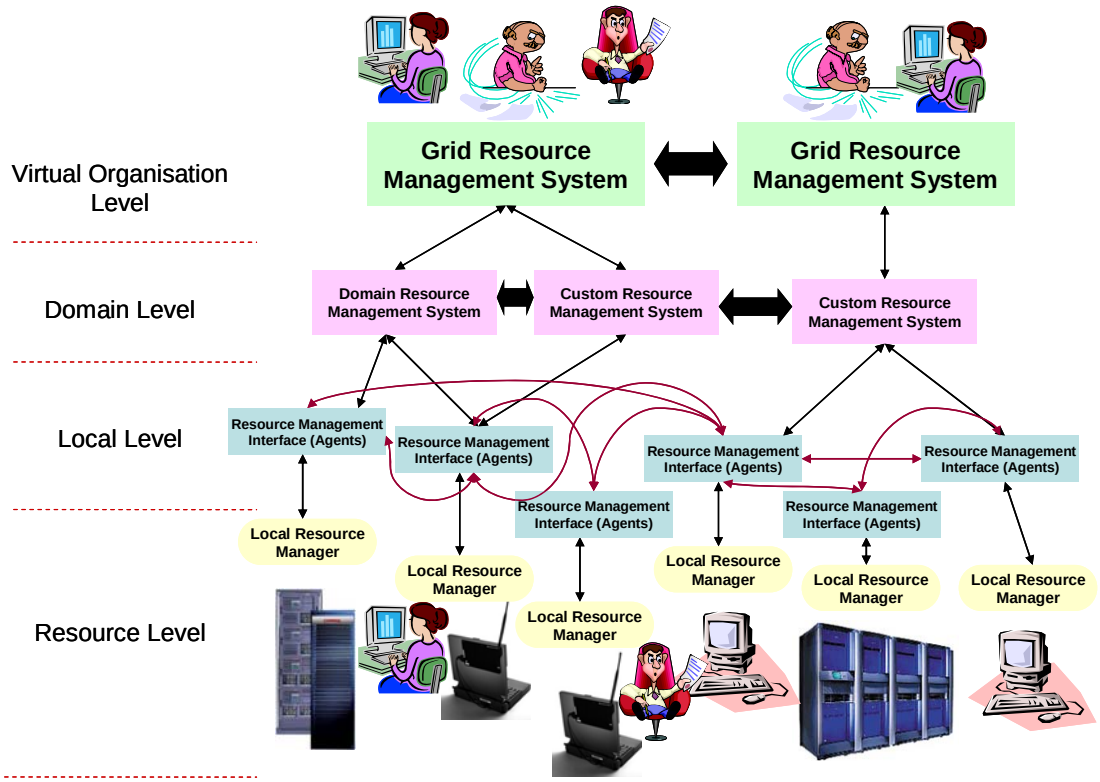

Figure 2.4 Hybrid Grid Resource Management system

Making hybrid management systems work presents some challenges, among them sorting out the contractual relationships and interdependencies between agents and hierarchical managers. These are complex architectures to design and implement. Hybrid Resource Management Systems have more media usability as compared to hierarchical resource management systems. They do have a notion of relative importance of real-time information of the managed entities (basically, resources). An important drawback is that final customers as well as resources owners have no high level and dynamic control over resource allocation. In our approach we have developed a hybrid solution in order to reach the proposed objectives of this system.

2.3.5 Classification of Current Grid Resource Management Architectures

In a nutshell, the presented analysis around these four solutions clearly shows that the most challenging methodology is the Hybrid Resource Management approach. It is also a very few exploited initiative. In Table 2.1, we are mapping the Grid Resource Management Systems of the most important current Grid Architectures with the type of Grid on their domain and the resource management approach. In just a few architectures the hybrid approach is adopted. The main reason is the low need of this type of solution on current projects. These projects have designed ad-hoc solutions for their resources infrastructure and then they are quite limited to only their environments. Although hybrid systems are better for GRM, implementation methodologies need to be improved. In future sections we will describe these methodologies and the main drawback of the current solutions.

TABLE 2.1. CLASSIFICATION OF CURRENT GRID RESOURCE MANAGEMENT ARCHITECTURES

Architecture	Grid Type	GRM Approach
AppLe [Ber97]	Distributed Computational Grid	Decentralized with heuristic resources estimation
DataGrid [Che00]	Data Grid	Hierarchical with predictive heuristic resources estimation
Condor [Tha05]	Distributed Computational Grid	Centralized and Hybrid monitoring instances
Globus [Bor05]	Data, High Performance and Distributed Computational Grid	Hierarchical
Javelin [Nea00]	Distributed Computational Grid	Decentralized
Legion [Cha99]	Distributed Computational Grid	Hierarchical
MOL [Buy01]	Distributed Computational Grid	Decentralized
NetSolve [Cas97]	High Performance Grid	Decentralized
Ninf [Nak99]	High Performance Grid	Decentralized
Nimrod-G [Buy00]	High Performance Grid	Decentralized
PUNCH [Kap99]H	High Performance Grid	Decentralized with predictive pricing models and rescheduling and Hybrid monitoring instances

2.4 Resource Management Functions in Large-scale Distributed Computational Grids

Fundamental to the success of a Grid is the ability to discover, allocate, negotiate, monitor, and manage the use of resources (computational and network-accessible capabilities) in order to achieve various end-to-end or global qualities of service, offer transparent services to final users in optimal times and balance the resource load along the full infrastructure. We must develop methods for managing resources/services across separately administered domains, with the resource heterogeneity, loss of absolute control, and inevitable differences in policies that result.

An ability to handle services requirements, resource discovery, resource monitoring, resource selection, job scheduling and job activation that influence computing performance are essential issues in Grid Computing. These activities are all related to one general concept, which is well known as the Grid Resource Management (GRM) process. Academic and industry researchers are working on better solutions to perform as efficiently as possible any of the before-mentioned activities in Grid Computing. The literature about GRM approaches is quite lengthy. Currently, we can find books [Nab04] [Fos03], taxonomy documents [Kra02], [Zan05] and obviously plenty of academic research papers proposing and explaining new methodologies and ideas to improve the GRM process.

Moreover, all this research is mainly supported by international projects that are normally financed by government institutes. The most active projects working on improving GRM approaches are:

- World Wide Grid (WWG) [http://www.gridcomputing.com/]
- NASA Information Power Grid (IPG)
- Micro Grid
- Alliance Grid Technologies
- EuroGrid
- NorduGrid
- TeraGrid
- GridCAT
- OGSA Config
- CRO-GRID Infrastructure
- EGEE
- World Community Grid

Despite this huge list of projects and research initiatives there are some issues unsolved in Grid Resource Management for Large-scale Computational Grids. In order to understand these issues, we have classified in our study the GRM functions in four phases: Grid Services Requirements, Grid Resource Discovery and Monitoring, Grid Resource Scheduling, and Jobs Allocation and Activation. In the following sub-sections we will describe the most important functions of these four GRM phases. Then we will highlight the most significant unsolved issues and finally we will explain why current proposals are not enough to properly cover the non-functional requirements[1] of the large-scale GRM.

2.4.1 Grid Services Specifications and Requirements

Two forces are leading the development of Grid Computing Services: The Globus Alliance, which oversees the Globus Toolkit [Bor05]; and the Global Grid Forum [GGF07], which is creating a set of open standards for Grid technologies and applications. The GGF includes academics, researchers, and small and big technology companies. The GGF's major efforts include the Open Grid Services Architecture (OGSA). The trend of OGSA is that every job running on the Grid no matter its complexity or simplicity is represented as a Service. This representation is currently known as Grid Services [Cza06].

Grid Services will be provided to users without any kind of distinctions related to network technology, operative platform and administrative domain. As a result, there have been significant increases in management complexity of the OGSA-based Grid Services provision, mainly because Grid Resource Management systems have to deal with the yet elusive rapid and autonomous creation, deployment, activation and management of the emerging Grid Services.

Grid Services incorporates Web Services [Ved02] standards to facilitate communication among heterogeneous resources. Grid Services expect that Web services mechanisms will become the interface for Grid Computing. These services are defined in terms of Web Services Description Language (WSDL) interfaces, and provide the mechanisms required for creating and composing sophisticated distributed systems. These mechanisms include code encapsulation and interface characterization, lifetime management, reliable remote invocation, credential management, and notification. Discovery services, like the Universal Description, Discovery, and Integration (UDDI) service, provide for locating services based on their

[1] Section 1.3 of the thesis defines these requirements.

functional characteristics, and then provide the detailed descriptions of the data types and interfaces needed to make use of those services.

Current Grid Service Customer's (GSCs) applications are becoming Grid Services when they are introduced into the Grid. Therefore, Grid Services are GSC's software, which is standardized through OGSA mechanisms. In these mechanisms it is possible to find detailed information about resources requirements to deploy and execute Grid Services. Moreover, GSCs will request certain Quality of Service (QoS) requirements for the requested Grid Services. These requirements should be satisfied by Grid Infrastructure Providers (GIPs). Otherwise, Grid Service Customer's (GSCs) Grid Services will not be neither deployed nor executed.

Therefore, three sources of Grid Service requirements, which will be considered as Grid Service constraints along this thesis, are showing up here: the OGSA-based constraints, QoS constraints and GIPs' capabilities constraints. Detailed information about these three sources of requirements is in Chapter 3.

Solutions for accomplishing Grid Services taking into account these three sources of requirements are needed. Current approaches have limitations, particularly related to high-bandwidth, network re-configuration, reliability, scalability, flexibility and persistence. Moreover, it is necessary that there be a major entity in charge of providing high-level management to allow quick and autonomous deployment, activation and reservation of Grid Services in a transparent way from the GSCs point of view. In this field, some projects appeared with proposals to improve the management of Grid Services, projects such Condor-G [Tha05], Control Architecture for Service Grids [Gra02], Data Grid [Che05] and Nimrod-G [Buy00]. Policy-based middleware systems for Grid Services were presented in [Yan02] and [Maga04], involving technologies as active networks and the GARA architecture but without any constrains regarding OGSA compatibility. Some of the above developments are just functional improvements within the context of their respective projects, whilst Condor-G and G-QoSM [Ali04] have the drawback of not being completely autonomous. Although G-QoSM is coping with similar features that are presented in this paper, it is not very flexible. It bases its reliability on a central component, the middleware Grid Resource Manager (GRM), which may present overload problems for large amounts of Grid Service requests [Cza06].

There are a number of Grid Middlewares that are available, and they implement mechanisms to manage (fulfil en lugar de manage) some of the mentioned Grid Services requirements. Globus Toolkit (GT4) [Bor05] is the most frequently used. It contains elements of all of the OGSA core service areas except for self-management. It will be used as the core service layer for the National Science Foundation TeraGrid project [Ter05]. It is also used in the

bioinformatics Grid GeneGrid [Kel05], and GridCast [Cas07], a Grid to support the delivery of multi-media for the BBC. Unfortunately, GT4 is too complex to implement in large-scale Grids.

gLite [Gli07] also implements mechanisms to integrate Grid Services Requirements. It supports research in high-energy physics. gLite is used in the "Enabling Grids for E-SciencE" EGEE project [Ege07] and the LHC Computing Grid [Lhc07]. Another large physics project is the Open Science Grid OSG [Osg07], which uses elements of GT4 and gLite. The Legion system, which is one of the oldest software platforms for Grid computing, is being redeveloped as a web SOA by the University of Virginia and is being used in the Global Bio Grid [Bio04].

Regardless of the huge amount of Grid Resource Management Systems available [Nab04], there are still missing some flexible architectures from the users' point of view which are able to collaborate with the other GRM phases (monitoring, scheduling and allocation), in order to offer transparent deploying of Grid Services into GIPs. Our approach goes one step further, offering the ability to handle the management requirements of grid services by means of the Policy-based Grid Resource Management System [Maga07a].

Policy-based Management is an excellent alternative solution that meets the presented sources of requirements of Grid Services. In Policy-based Management approaches, the support for distribution, automation and dynamic adaptation of the behaviour of the managed system is achieved by using policies [Ver00]. The main benefits of using Policy-based technology are improved scalability and flexibility for the management system. Scalability is improved by uniformly applying the same policy to large sets of devices and objects, avoiding the strenuous task of re-coding. Moreover, the decision-making ability that traditionally existed on centralized (more intelligent) management entities is embedded in each policy. Hence, there is a severe reduction in the management information that may propagate in the network since the intelligence now exists in the point where it is needed. Flexibility is achieved by separating the policy from the implementation of the managed system. Policy can be changed dynamically, thus changing the behaviour and strategy of a system, without modifying its implementation or interrupting its operation. Separating the policy from the manager entities also enhances the flexibility. Although the implementation of the managers may alter, the management principles remain untouched, as they were originally defined in the policies.

2.4.2 Grid Resource Discovery and Monitoring

Grid Resource Discovery and Monitoring involves analysing which resources are available on the Grid Infrastructure. It is an important aspect of the overall efficient usage and control of the Grid. The Grid resource discovery and monitoring should be able to monitor any kind of resources of interest that can include all manner of network devices, sensors, computing

servers, storage servers, software and applications. The Grid requires that a broad range of data be monitored and collected for a huge variety of applications that users are demanding.

The most complex activity in this area is to develop heterogeneous mechanisms to monitor and discover any kind of shared resources. We obviously are assuming that monitoring will be done based on a highly distributed architecture because the amount of resources is too big to be controlled by centralized mechanisms. We could classify in three methodologies any of the existing Grid Resource Discovering and Monitoring systems. These methodologies are: Pull, Push and Hybrid.

The Push methodology is simply pushing data from itself to the monitoring system. This can be done periodically, or by request from the monitor system asynchronously. The advantage of this methodology is that the monitoring system's load can be reduced to simply accepting and storing data, and it does not have to worry about timeouts for communication calls, parsing Operating System (OS) specific call results, etc. The disadvantage of this mode is that the logic for the polling cycle/options are not centralized at the monitoring system, but distributed to each remote node. Thus changes to the monitoring logic must be pushed out to each node.

On the Pull methodology, one or more processes in the system actually poll the system elements in some thread. During the loop, devices are polled via agents' calls (e.g. SNMP, IP SLA), resources can be accessed via remote communication protocols such as telnet and SSH to execute scripts or dump files or execute other OS-specific commands, applications can be polled for state data, or their state-output-files can be dumped. The advantage of this mode is that there is little impact on the host/device being polled. The host's CPU is loaded only during the poll, and the rest of the time the monitoring function plays no part in CPU loading. The main disadvantage of this mode is that the monitoring process can only do so much in its time, and if polling takes too long, the intended poll-period gets elongated. An intermediate methodology between pull and push is the hybrid approach, where the System Configuration determines where monitoring occurs, either in the monitoring system or agent. This is especially useful when setting up a monitoring infrastructure for the first time, and not all monitoring mechanisms have been implemented.

Grid Resource Discovery and Monitoring yields two main classes of data: current state information and event reporting (e. g. failure notifications). State information is best suited to be delivered using the pull model, meaning that an entity interested in the data must actively ask for it. Event information however is best delivered using the push model where the component that generated the event arranges for a notification to be sent to all interested parties. Therefore it is desirable for a monitoring service to support both push and pull data delivery models.

In order to evaluate how functional and efficient a Grid Resource Discovery and Monitoring System in large-scale Computational Grids is, there are some essential features that it should take into account, which are independent of the methodology implemented (push, pull, hybrid). These features are: Non-Intrusively, Heterogeneity, Flexibility and Scalability. Basically, the resource monitoring and discovering system should be minimal intrusively in the monitored resources. Otherwise, the monitored information will not be the real one because of the resources consumed by the monitoring system. Grid Resource Discovery and Monitoring also should be as more heterogeneous as possible. It means that resources from different technologies, manufactures and platforms have to be monitored at any time without specific configuration. We are aware that is almost impossible to build a system able to identify any kind of resources and starts properly process to monitoring their behaviour but it is worthwhile to design novel systems to cope with the resources heterogeneity.

Obviously, scalability is essential in the resource monitoring system. A non-scalable design is not functional in large-scale Grids. Agents-based and P2P-based approaches are the most common solutions. Finally, resource-monitoring systems need flexible mechanisms to modify their “polling periods”. The polling periods are the intervals of time between every measure from the monitoring system to the resources in order to get the resources availability information. It is an important feature because most of the current monitoring approaches for large-scale Grids are based on non-flexible polling periods [Mas04] and [Bak05]. This is a drawback on these systems. With fixed polling periods, the monitoring methodology will be consuming more resources because is not smart enough to decrease its polling periods when the resources are in idle. Moreover, the fixed methodology could offer false information when resources have highly activity. Therefore, dynamic monitoring is a great solution. The complication on this alternative is the programming of the methodology. The algorithms to decide when decrease or increase the polling periods are quite complex to design. Therefore, revolutionary designs are needed in this area to offer flexible Grid resource-monitoring systems.

Grid Resource Discovering and Monitoring systems could report instantaneous or statistical resources availability information. We are also assuming that any Grid Resource Discovering and monitoring system for Large-scale Computational Grids guaranteed real-time information. Monitoring systems failing in offering real-time information are not functional for this type of Grid.

- **Instantaneous:** This approach offers resource availability information constantly in real-time every determinate interval of time (polling period).

- **Statistical:** In this approach, the monitoring system is not offering just snapshot information of the resources availability. In this case, the monitoring system is also offering an average of the resource availability information. This methodology is currently used for post-analysis of the resources performance more than pre-scheduling analysis.

Unfortunately, none of the current resource schedulers are supporting their scheduling decisions based on statistical information [Nab04]. Here we have a great research area to offer better resource availability information for scheduling proposes. The disadvantage for this kind of monitoring systems is that they need more storage capacity and stronger analysis algorithms for the statistical information. In a flexible solution, as we are proposing in this thesis, the algorithm's analysis could be simplified because of the distribution of the sources of information.

We believe that monitoring activity should not be specifically designed for any technology (Grid, P2P, Specific Distributed Systems, etc). The following approaches fail in fulfilling these couple of conditions. We will present an alternative solution that we have considered as a good alternative for those Grid Resource Management Systems which monitoring times or methodologies are not as good as their developers could desire.

A lot of work has been done in Grid Monitoring. Due to space constraints, we cannot discuss all of them, but definitely the best summary so far, is presented in [Zan05]. Currently, SNMP-based monitoring agents have been implemented by many researchers to solve different problems [Pav04] [Sub00]; each system has its own strengths and weaknesses. However, for resource monitoring and management, we believe that none fulfil the criteria that are desired in emerging generation networks [Sch03]. GridLab [GridLab] aims to enable applications to fully exploit dynamically changing computational resources. In order to accomplish this, a variety of mechanisms have been developed such as notifications about changes of job states, complex workflow support, multi-criteria user-preference driven and prediction-based selection of the best resources, submission of jobs with time constraints, and many others. The capability of a dynamic adaptation has been achieved using job migration and pointing techniques.

One of the most similar approaches to SBLOMARS has been presented in [Bak05]. GridRM is also a generic monitoring architecture that has been specifically designed for the Grid. It was developed joining several technologies and standards like Java (applets, servlets and JDBC), SQL Databases, Grid Monitoring Architecture and it follows several recommendations by Open Grid Forum. SBLOMARS could be more competitive base on its low resources consumption and its availability to offer at any moment reliable information regarding

availability of computational resources. GridRM uses the Mercury grid monitoring system [Bal03] to deliver trace data to the host of visualization and PROVE [Bal01] visualizes trace information on-line during the execution of the grid applications. Unfortunately, they are oriented to monitor application performance instead of resources monitoring and their two fixed layers structure is a scalability lack.

MonAlisa [Leg04] is a distributed monitoring service based on JINI/JAVA and WSDL/SOAP technologies, which provides monitoring information from large and distributed systems to higher level services that require such information. It could be argued that JINI is using multicast, which is not always available, and places scalability limits. Although, MonAlisa is a well justified flexible system, it runs remote scripts in Grid resources get behavioural information, this mainly causes an extra traffic in the network.

The Monitoring and Discovery Service (MDS), constitutes the information infrastructure of the Globus Toolkit [Bor05]. Globus was designed and implemented as part of the Open Grid Services Architecture (OGSA) [Fos02a]. The Lightweight Directory Access Protocol (LDAP) [Ldap07] is adopted as a data model and representation, a query language and a transport protocol for MDS. However, LDAP features a non-declarative query interface that requires knowledge of the employed schema. In addition, the performance of OpenLDAP's update operation, which is by far the most frequently used, has been very much criticized.

Many other approaches could be found on the literature. NetLogger [Tie02] is both a methodology for analyzing distributed systems, and a set of tools to help implement the methodology. It provides tools for distributed application performance monitoring and analysis. Remos [Dew04] aims to allow network-aware applications to obtain relevant information about their execution environment. It defines a uniform heterogeneous interface that addresses statistical information and efficiency for these environments. Finally, Java Agents for Monitoring and Management [Law00] is a distributed set of sensors that collect and publish monitoring information regarding computational systems. These systems have been presented for different frameworks and some of them are not available any more.

There are also alternative solutions based on heuristic methodologies to improve real-time monitoring mechanism. A-GAP [Gon06] addresses the problem of continuous monitoring in distributed systems. It is a generic aggregation protocol with controllable accuracy objectives for large-scale network environments. A-GAP allows for continuously computing aggregates of local variables by creating and maintaining a self-stabilizing spanning tree and incrementally aggregating the variables along the tree. A-GAP can reduce the overhead significantly when allowed a modest error in estimating an aggregate. Heuristic solutions in the monitoring

functions are functional when they are independent systems because the minimal error could drastically deteriorate the resource selection phase in the Grid Resource Management process.

2.4.3 Grid Resource Scheduling

Jarek Nabryski et al. define grid resource scheduling, as the process of making scheduling decisions involving resources over multiple administrative domains [Nab04]. This process includes searching multiple administrative domains to use available resources from the Grid Infrastructure in order to satisfy Grid Services hard and soft constraints demanded by Grid Services Customers (GSCs).

Therefore, scheduling is the fact of assigning available resources on the Grid Infrastructure for requested Grid Services. A Grid Service could be scheduled on a single resource or a set of them at a single site or multiple sites. Grid Services are anything that needs a resource, from a bandwidth request to an application or to a set of applications. The term resource means anything that can be scheduled: a machine, the disk space, a QoS network, and so forth.

Grid Services require the coordinated processing of complex workflows, which includes scheduling of heterogeneous resources within different administrative domains. A typical scenario is the coordinated scheduling of computational resources in conjunction with data, storage, network and other available Grid resources, like software licenses, experimental devices, etc. The Grid scheduler should be able to coordinate and plan the workflow execution. That is, it should reserve the required resources and create a complete schedule for the whole workflow in advance.

Jobs submitted to Grid resource schedulers are evaluated based on their service constraints and then allocated to the respective resources for execution. This will involve complex workflow management and data movement activities to occur on a regular basis. In order to make the best use of an available resource, part or all of the resources may have to be reserved in advance. Depending on the resource, an advance reservation can be easy or hard to do and may be done with mechanical means or human means. Moreover, the reservations may or may not expire with or without cost.

We have classified current resource schedulers as static and dynamic approaches. This choice indicates the time at which the scheduling decisions are made. In the case of static scheduling, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. By contrast, in the case of dynamic scheduling, the basic idea is to perform task allocation on the fly as the application executes. This is useful when it is impossible to determine the execution time,

direction of branches and number of iterations in a loop as well as in the case where jobs arrive in a real-time mode.

Current Grid Resource Scheduling systems only support interactive jobs by trying to start them immediately and returning an error if that is not possible. A better approach would be to schedule the job as normal (probably with some time constraints, like it should be started only during work hours). Before really starting the job the broker could then check if the user is on-line at the moment, and postpone the job if the user is not available.

Due to the fact that dynamic scheduling is usually applied when it is difficult to estimate the cost of jobs are coming online dynamically, which is actually the case in our solution, A good example of these scenarios is the job queue management in some scheduling systems like Condor [Tha05] and Legion [Cha99]. Since the cost for an assignment is not available, a natural way to keep the whole system health is balancing the loads of all resources. The advantage of dynamic load balancing over static scheduling is that the system need not be aware of the run-time behaviour of the application before execution. It is particularly useful in a system where the primary performance goal is maximizing resource utilization, rather than minimizing runtime for individual jobs. According to how the dynamic load balancing is achieved, there are four basic approaches [Fan06] [Nab04]:

I. **Unconstrained:** In the unconstrained approach, the resource with the currently shortest waiting queue or the smallest waiting queue time is selected for the incoming task. This policy is also called Opportunistic Load Balancing (OLB) [Sak04] or myopic algorithm. The major advantage of this approach is its simplicity, but it is often far from optimal. An unconstrained solution is Condor-G. It allows the user to treat the Grid as a local resource. The same command-line tools perform basic job management such as submitting a job, indicating executable input and output files and arguments, querying a job status or cancelling a job. Condor-G can cooperate with the following middleware: Globus Toolkit 4 [Bor05], Unicore [Ram02] and NorduGrid [Nor07], and it can submit jobs to Condor-G [Tha05], PBS and Grid Engine (SGE / N1GE) [Kis05] scheduling systems. Therefore, there are not constrained related to the Grid Service involved in this scheduling systems because user is controlling directly the resources assigned by the scheduler.

II. **Balance-constrained:** The balance-constrained approach attempts to rebalance the loads on all resources by periodically shifting waiting tasks from one waiting queue to another. In a massive system such as the Grid, this could be very costly

due to the considerable communication delay. So some adaptive local rebalancing heuristic must be applied. For example, tasks are initially distributed to all resources, and then, instead of computing the global rebalance, the rebalancing only happens inside a "neighborhood" where all resources are better interconnected. This approach has several advantages: the initial loads can be quickly distributed to all resources and started quickly; the rebalancing process is distributed and scalable; and the communication delay of rebalancing can be reduced since task shifting only happens among the resources that are "close" to each other. A scheduling algorithm of this type is proposed and evaluated in [Chen02].

III. **Cost-constrained:** An improved approach to balance-constrained scheduling is cost-constrained scheduling. This approach not only considers the balance among resources but also the communication cost between tasks, this cost is the latency between jobs from the same Grid Service running in distributed nodes. Instead of a blind job exchange, jobs will be checked to determine their mutual communication costs. If the communication cost brought by a task shift is greater than the decrease in execution time, the task will not move; otherwise, it will be moved. This approach is more efficient than the previous one when the communication costs among resources are heterogeneous and the communication cost to execute the application is the main consideration. It is also flexible, and can be used with other cost factors such as seeking lowest memory size or lowest disc drive activity, and so on. In [Kur04], authors propose a scheduling policy based on these concepts, but the objective of the rescheduling phase in their case is not for load balancing and cost optimizing, but rather for guarantee certain efficiency deploying the Grid Services into the Grid. An example of cost-constrained load-balancing mechanism is the WMS scheduler. It is based on a metascheduler developed in LHC Computing Grid (LCG) and submits jobs to computational grids through Condor-G. Therefore the functionality of job management and its interoperability is almost the same as in the case of Condor-G. There are two approaches to job scheduling. The first one is based on the usage of Condor-G as an intermediary that pushes jobs into computational units. The second one, called the "pull model", expects the computational grid to take a job from a queue and schedule it for execution.

IV. **Heuristic-constrained:** The challenge in Grid resource scheduling is to reduce the time of completing the process. In fact, the scheduling problem in dynamic and large-scale Grids is a typical NP-hard problem, which in practice has to be solved by heuristics methodologies. Actually, some proposals were designed for large-scale distributed computing Grids [Gar00] but some of them fail in real environments. Many heuristic algorithms have been proposed to deal with specific cases of job scheduling but they fail, or behave inefficient, when applied to other problem domains. An important family of these heuristic solutions is based on Genetic Algorithms (GAs), which apply evolutionary strategies to allow for a faster exploration of the search space. GAs have proven to be successful in getting better distribution of the jobs through the entire network [Zom01]. Many researchers have investigated GAs to schedule tasks in homogeneous [Ahm01] and heterogeneous [Pag05] multi-processor systems with remarkable success.

Generally, approaches to resource scheduling depend strongly on the architecture of the whole system, which, in turn, depends on the underlying middleware used. On one hand, the most of the current architectures are mainly based on Globus Toolkit 4.0 and UNICORE systems. On the other hand, the remaining solutions presented are Service-Oriented Architectures (SoA) [Tho05]. The main difference between Globus Toolkit and UNICORE is that the former is more “horizontal” than the latter in terms of their functionality. On the contrary, in the service-oriented approaches, there are a group of services that can communicate with each other instead of a monolithic application. All approaches give additional functionality but also cause some difficulties. Globus provides a wide set of low level mechanisms that contain many tools needed for efficient resource management and scheduling, such as an information system (MDS), the resource allocation manager (GRAM), and so on. In contrast, there are many reservations concerning Globus feasibly. It does not provide high-level mechanisms such as management of workflows and so forth. UNICORE provides high-level tools and high security standards.

Challenging issues from a grid resource scheduling perspective include the need to co-schedule what are often scarce and expensive resources, the scalability of protocols and algorithms to tens or hundreds of thousands of nodes, latency-tolerant algorithms, and achieving and maintaining high levels of performance across heterogeneous systems. Actually, we will demonstrate in Chapter 6 that these contributions are reached in our solution.

2.4.4 Jobs Allocation and Activation

The Jobs Allocation and Activation are the final activities on the Grid Resource Management process. It provides the user to access the Grid resources in order to run and terminate jobs remotely. The queued job is sent to the selected resource through standard interfaces such as: Globus Toolkit, Condor, and Linux CLI. Once the job has been assigned, these interfaces create a job manager instance to handle the job process for future information (status and results).

The job allocation activity consists of transferring the data or information that should be processed to the node or nodes selected by the Grid Resource Scheduling activity. On the other hand, the job activation activity consists of executing the corresponding code and offering the adequate mechanisms to get the result back to the appropriate management instance. For instance, let's assume the problem of calculating the inverse of a 25 x 50 Matrix M. Therefore, the job allocation activity sends the matrix data to the selected node (only one node in this example) and stores this information on the node's buffer memory. The job activation activity executes the operations matrix software (Grid Service) and selects the inverse operation of the Matrix M. The resulting 50 x 25 Matrix M' is sent back to the resource manager instance in charge of this job.

Job Allocation and Execution may be as easy as running a single command or as complicated as running a series of scripts and may or may not include setup or staging. When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines will include an e-mail notification parameter. For fault-tolerant reasons, however, such notification can prove surprisingly difficult. Moreover, with so many interacting systems one can easily envision situations in which a completion state cannot be reached. And of course, end-to-end performance monitoring to ensure job completion is a very open research question.

Some systems require users to manage intermediate data transfer in the allocation process, rather than providing automatic mechanisms to transfer intermediate data. We categorize approaches of automatic intermediate jobs allocation into centralized, mediated and peer-to-peer [Yia06].

- **Centralized:** Basically the centralized approach transfers intermediate data between resources via a central point. For example, the central workflow activation engine can collect the activation results after task completion and transfer them to the processing entities of corresponding successors. The centralized approach is easy to implement and suits workflow applications in which large-scale data flow is not required.

- **Mediated:** Rather than using a central point for the mediated approach, the locations of the intermediate data are managed by a distributed data management system. For example, in the Pegasus system, the intermediate data generated at every step are registered in a replication catalog service [Ven06], so that input files of every task can be obtained by querying the replication catalog service. The mediated approach is more scalable and suitable for applications that need to keep intermediate data for later use. Jobs allocation and activation systems designed under Policy-based Management technology are classified under this category. Although, they could appear quite centralized approach, their hierarchical design supports high levels of scalability. Moreover, the distribution of their components also improves the flexibility and scalability of the full architecture.

- **Peer-To-Peer:** This approach transfers data directly between processing resources. Since data is transmitted from the source resource to the destination resource without involving any third-party service, it significantly saves the transmission time and reduces the bottleneck problem caused by the centralized and mediated approaches. Thus, it is suitable for large-scale intermediate data transfers. However, there are more difficulties in deployment because it requires a Grid node to be capable of providing data management services. In contrast, centralized and meditated approaches are more suitably used in applications such as bio-applications, in which users need to monitor and browse intermediate results. In addition, they also need to record them for future verification purposes

In large-scale Distributed Computational Grids, the simple act of submitting a job can be made very complicated by the lack of standards. Some systems, such as the Globus GRAM approach [Cza98, Glo07a], wrap local scheduling submissions but rely heavily on local-parameter fields. Ongoing efforts in the Global Grid Forum [GGF07, Glo07b] address the need for common APIs [Glo07c]. Most often, a user will run scp, ftp or a large file transfer protocol such as GridFTP [All02] to ensure that the data files needed are in place. In a Grid setting, authorization issues, such as having different user names at different sites or storage locations, as well as scalability issues, can complicate this process.

In comparison with before-mentioned Grid Job Allocation and Activation management approaches and methodologies, Policy-based Management (PbM) could be introduced again to solve many of the Grid Job Allocation and Execution issues. PbM offers a more flexible and

customisable management solution allowing each resource to be configured on the fly, for a specific application tailored for a customer. Policy-based technology allows for enhancing the management architecture dynamically, for the introduction of new applications or device specific policies, tailored to realise complex tasks, and for the automation of network management tasks. Therefore, Policy-based solutions should be highly flexible. They also have to offer mechanisms to auto-extend its management activities, which are normally named as action and conditions interpreters. The Policy-based Grid Resource Management architecture described in Chapter 3 shows how this approach can be used to overcome many management problems inherent in the Grid Jobs Allocation and Activation phase.

2.5 CONCLUSIONS

Grid Resource Management (GRM) is regarded as a vital component of the Grid infrastructure. Its main function is to coordinate and share multiple kinds of resources efficiently; and, in the above-mentioned approaches, GRM faces several challenges that make the implementation of practical systems a very difficult problem. Among these challenges we would like to emphasize the fact that GRM systems must fulfil strict functional requirements from heterogeneous, and sometimes conflicting, domains (e.g., the users', applications and networks domains). In addition, GRM systems must adhere to non-functional requirements that are also rigid, such as reliability and efficiency in terms of time consumption and load on the host nodes.

Grid Resource Management problem has been solved by means of centralized, decentralized, hierarchical and sometimes hybrid approaches. On one hand, centralized and hierarchical approaches fail when the number of resources increases or when resources maintain certain mobility in the Grid. On the other hand, decentralized approaches are considered better solutions mainly for heterogeneous networks, no matter the level of resource mobility presented. The most important disadvantage is the complexity of their implementation. Thus, a level of fusion is required to improve efficiency and functionality in current approaches. Therefore, hybrid systems are everyday getting more attention from both communities, academic and commercial.

An opportunity to improve current GRM systems is found in the requirements of management of Grid Services. Current solutions do not take into account the all three sources of Grid Services requirements explained in this Chapter. Some of these solutions are Service Oriented Architectures (SOA) [Tho05] and Grid Services requirements from OGSA standard are involved in the scheduling decisions but GSCs requirements are not always taken into account or GIPs are deprecated because of the assumption of resources are always available.

Current Resource Discovering and Monitoring systems are not flexible. Most of them are based on fixed polling periods. We believe that resources consumed by the monitoring system could be reduced if the polling periods are allowed to be auto-adjustable. Moreover, it is possible to reduce the number of detection change failures because a flexible monitoring system could reduce its polling periods and then to get more information about resource availability.

Current GRM systems research is focused mostly on solving either one or the other of the two main problems at hand. The first of these is concerned with evenly balancing resource loads throughout a Grid; and, in this area, plenty of researchers have proposed a wide range of technologies with varying degrees of success. The second problem, however, involves shortening the makespan. The makespan of a schedule is the time required for all jobs to be

processed by a set of unknown resources. For quite some time, a comprehensive solution addressing both of these problems proved elusive. In other words, a load-balanced solution with a minimal makespan was necessary. The approach we present in this thesis provides such a solution.

In order to achieve an optimal solution between these two problems, we believe that monitoring activity should be decentralized and allocated on the Grid Infrastructure (resources available) and therefore should be almost unperceived for the hosting node and should not be specifically designed for any technology (Grid, P2P, Specific Distributed Systems, etc). GRM approaches presented in this Chapter fail in fulfilling at least one of these two conditions. Our research in this thesis was driven by this aim and we finally concluded with an alternative to current GRMS that represents a step ahead of the state of the art.

Chapter 3

POLICY-BASED GRID RESOURCE MANAGEMENT ARCHITECTURE

3.1 INTRODUCTION

Grid Computing promises to change the way organizations tackle complex computational problems. Nevertheless, Grid Computing is an evolving area, and standards and technology are still being developed to enable this new paradigm. The sharing of resources inherent in Grid networks introduces challenging resource management problems due to the fact that many applications need to meet stringent end-to-end performance requirements across multiple computational resources (memory, processor, storage, etc). Furthermore, they should also guarantee fault tolerance and network level Quality-of-Service (QoS) parameters, as well as arbitrate conflicting demands.

From the users point of view, Grids should be completely transparent. They just need to send the information data to be processed by a specific application and receive their results based on Service Levels Agreements. Grid Service customers specify applications by means of Grid Services.

In this chapter we will show that Grid Services are becoming more popular in the Grid Community. Grid Services are the standard protocol and mechanism to completely automate the Grid Resource Management Process. For this reason a new Grid Computing generation is emerging, where schema conversion technologies via common meta-models and ontologies are required to allow data to be moved between storage resources and be shared between tools with different data format requirements. These novel schemas and models are based on on-line libraries for conversion and mapping, to be used in a variety of scientific, social and business domains.

At the same time, there are other innovations with significant impact on Grid Computing, like the creation of mechanisms to handle a large number of heterogeneous resources (computing, storage, networks, services, applications, etc.), the adequate use of data from a wide variety of sources and technical domains, the necessity of designing and implementing new techniques to allow for the interoperability between different data sources is constantly growing. One of the key challenges will be the co-ordination and orchestration of resources to solve a particular problem or perform a business process.

This chapter explains why the current generation of Grid Resource Management systems relies heavily on the program designer or the user to express their requirements in terms of resource usage. Such requirements are usually hard-coded in a program using low-level primitives. But the Grid needs to handle resources in a more dynamic way. In other words, grid applications will require the co-ordination and orchestration of grid elements at run time. Two of the most important grid features are management autonomy and fault-tolerance. These require redundant resources (computation, data base, network), and also an autonomous and self-regulatory model that ensures the proper working of the management architecture by itself. Flexibility and scalability are crucial; solutions should be able to support Grids consisting of billions of nodes. This requirement immediately stresses the importance of finding an appropriate trade-off between Grid Services Customers (GSCs), Grid Services Repositories, and Grid Infrastructure Providers (GIPs).

This chapter describes our proposed Policy-based Management Architecture which is in charge of controlling the communication workflow of the Grid Resource Management process. The components and interfaces of this architecture are detailed explained as well as their main functionalities.

The structure of this chapter is as follows: In section 3.2 we briefly describe Police-based Management technology. Section 3.3 is an overview of our Policy-based Management proposal for solving the Grid Services Management and Jobs Allocation and Activation phases. We explain the deployment and activation process of Grid Services using our proposal in Section 3.4. Finally, Section 3.5 concludes this chapter.

3.2 POLICY-BASED MANAGEMENT SYSTEMS

Policy-based Management (PbM) [Stra03] is a very suitable paradigm to manage complex heterogeneous environments, such as Grid Computing (Figure 3.1). The Policy-based Grid Resource Management Architecture (PbGRMA) presented hereafter, is a realization of this paradigm.

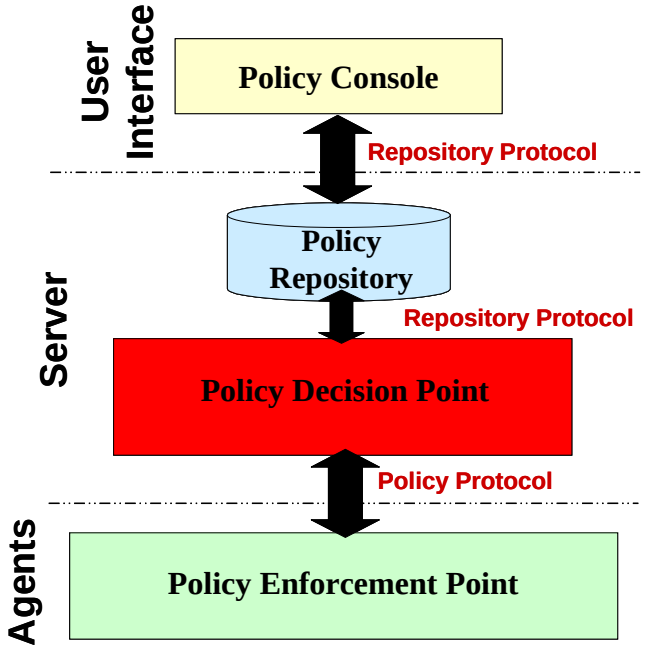

Figure 3.1 Core architecture of policy-based management technology

In policy-based management approaches, support for distribution, automation, and dynamic adaptation of the behaviour of the managed system is achieved using policies. *"Policies are derived from the goals of management and define the desired behaviour of distributed heterogeneous systems and networks"* [Ver01]. The main benefits of Policy-based technology are improved scalability and flexibility for the management system.

Scalability is improved by uniformly applying the same policy to large sets of devices and objects, avoiding the strenuous task of re-coding. Moreover, the decision making ability that traditionally existed on centralized management entities, is embedded in each policy. Hence, there is a severe reduction of the management information that may propagate in the network, since the intelligence now exists in the point where it is needed.

Flexibility is achieved by separating the policy from the implementation of the managed system. Policies can be changed dynamically, thus changing the behaviour and strategy of a system, without modifying its implementation or interrupting its operation. Flexibility also is enhanced by separating the policy from the management entities. Although the implementation of the managers may change, the management principles remain untouched, as they were originally defined in the policies.

Current Policy-based solutions for accomplishing Grid Services management requirements have limitations, particularly related to high-bandwidth, network re-configuration, fault-tolerance, reliability, scalability, flexibility and persistence. In this chapter we will attempt to give an explanation of the methodology implemented in our approach to solve most of these limitations.

The Policy-based Grid Resource Management Architecture (PbGRMA) presented in this chapter covers two of the phases of the Grid Resource Management process; namely, Grid Services Management, and Jobs Allocation and Activation. In the first phase, our approach goes one step further, offering the possibility of handling hard and soft constraints involving Grid Services Management. Hard constraints are specific applications and resource requirements that must be fulfilled in order to run a job that belongs to a Grid Service. Soft constraints are management policies for resource utilization, deadlines, and response times. In the second phase, we will obtain a reliable Jobs Allocation and Activation system by means of an extension of our previous Policy-based Management System (FAIN-PbMS) [Sal03]. We have extended the dynamic components and interfaces (Policy Decision Points – PDPs and Policy Enforcement Points PEPs) of the FAIN-PbMS to make it compatible with Globus Toolkit Middleware and then to allocate Grid Services along large-scale Grid Infrastructures.

One the most important contributions of our Policy-based Grid Resource Management Architecture design is that it simplifies communication between Grid Services Customers (GSCs), Grid Services Repositories, and Grid Infrastructure Providers (GIPs). It is designed to retrieve the Grid Service handler from the UDDI Registry, and the Grid Service requirements from the Grid Service Descriptor Server in name of the GSCs. In Figure 3.2 we illustrate the business model evolution that we are proposing in our solution in order to simplify Grid Services deployment in large-scale Grids.

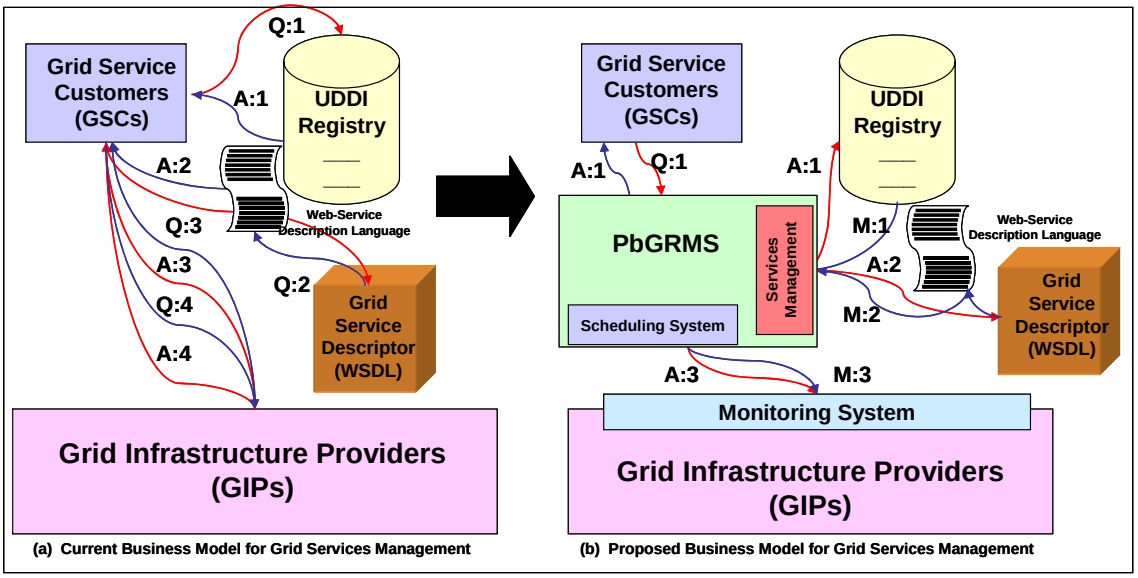

Figure 3.2 Evolution of the business model proposed for Grid Services Management

3.3 Overview of The Policy-based Resource Management System in Large-scale Computational Grids

In Chapter 1 we have defined Grid Resource Management activity as a process with four main phases: Grid Services Management (GSM), Grid Resource Discovery and Monitoring (GRDM), Grid Resource Scheduling (GRS), and Jobs Allocation and Activation (JAA). The PbGRMA is the structure of the proposed Grid Resource Manager system. It is in charge of the GSM and JAA phases. However, the PbGRMA is also somehow controlling the monitoring system and the scheduling system. Although, these two components are independent from the PbGRMA, they need to be part of a higher level architecture that would be sending them requests of Grid Services. In section 3.4, we describe the interactions between the PbGRMA with the monitoring system (SBLOMARS) and the scheduling system (BLOMERS).

The proposed Policy-based Grid Resource Management Architecture (PbGRMA) [Maga07a] is an extension to the Grid Computing environment of our previously conceived architecture [Sal03].. Although applicable to any user profiles, the system is essentially intended for non-massive Grid Service Customers (GSCs) accessing large amounts of computing, software, memory, storage, and even network resources. The PbGRMA deals with three different sources of resource requirements: users' QoS needs, Grid Infrastructure Providers' (GIP) resource availability information, and Grid Services Repositories' (GSRs) specifications set by both standards, Web Services – Resource Framework (WS-RF) [Cza06] and Open Grid Services Architecture (OGSA) [Fos02a].

The PbGRMA is designed as a hierarchically distributed architecture consisting of two levels: the Grid-Domain Management System (GDMS) and the Grid-Node Management System (GNMS). This hierarchical approach is shown in Figure 3.3. These proposed hierarchical levels combine the benefits of management automation with a reduction in management traffic and the distribution of management monitoring activities. The GDMS is in charge of managing resources from dispersed domains, which are commonly recognized as Virtual Organizations (VOs). The GNMS is in charge of managing resources inside the domain. It could be the case that more than one GNMS is used in one domain. This case is justified when the amount of resources in one domain is quite large. This capability assures a high level of scalability.

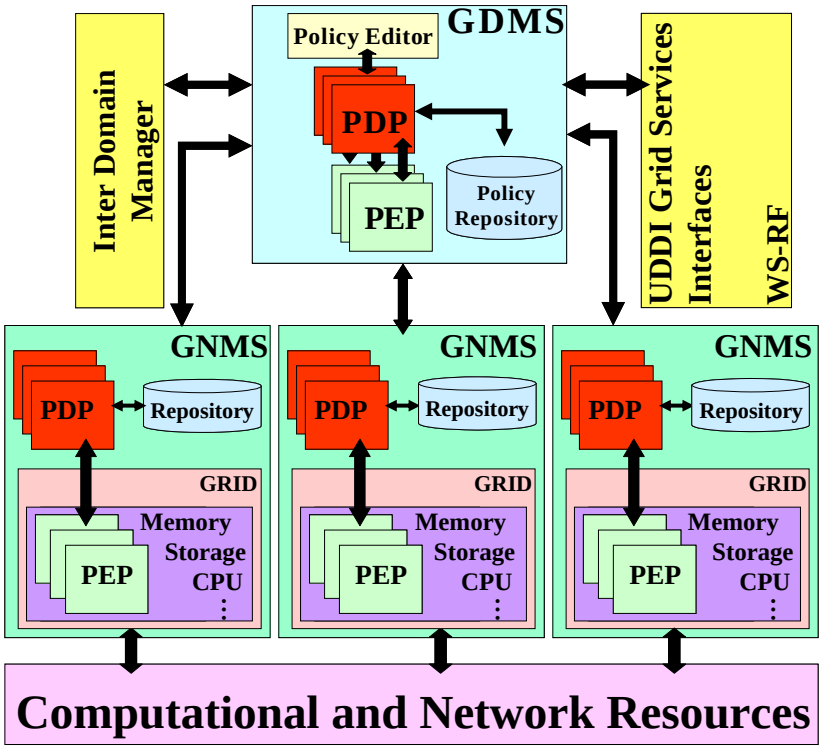

Figure 3.3 The hierarchical structure of the proposed Policy-based Grid Resource Management Architecture (PbGRMA)

This architecture is also capable of managing computational resources in order to balance resource exploitation along the Grid. It reaches a high level of scalability by extending itself its management components and policies interpreters needed to control multiple infrastructures regardless of network technology, operative platform, or administrative domain.

In the following sub-sections we will describe the design and architecture details of our proposed approach. We also include a sub-section where we explain the format of the Grid Service Policies in order to offer standard interfaces to external components such as the Inter-Domain Management system.

3.3.1 Grid Services Requirements

Grid Services are basically Web Services [Ved02] with improved characteristics and features. Normally, the customer must first invoke a service name registry, then request a service instantiation, and finally receive the Grid Service handler. These interactions are shown in Figure 3.2(a). Nevertheless, this approach involves many interactions from customers, and the most critical is that they have to find available resources by themselves. One of the most

important contributions of our Policy-based Grid Resource Management Architecture design is that it simplifies communication between Grid Services Customers (GSCs), Grid Services Repositories, and Grid Infrastructure Providers (GIPs). It is designed to retrieve the Grid Service handler from the UDDI Registry and the Grid Service requirements from the Grid Service Descriptor Server in name of the GSCs. Figure 3.2(b) illustrates the business model evolution that we are proposing in order to simplify the Grid Services deployment in large-scale Grids.

Therefore, PbGRMA deals with three different sources of resource requirements which will be considered as Grid Service constraints in this thesis:

I. **Quality of Service (QoS) Constraints:** In our approach, Grid Service Consumers (GSCs) can express QoS needs per service to deploy within the Grid Infrastructure. These requirements are introduced through a Graphical User Interface (GUI) within the Policy Editor. Also service-level management applications, according to the TMN layered structure [TMN00], can introduce policy information through an API given by the Policy Editor component. This allows the framework to be part of a complete stack of management services.

II. **Computational and Network Resources Constraints:** Resource availability is the amount of computational (cpu, memory, storage and application) and network (estimated bandwidth between nodes executing jobs from the same service) resources that are free to execute jobs forming Grid Services. This mechanism allows Grid Infrastructure Providers (GIPs) to balance the computing and networking load along the virtual organization.

III. **Open Grid Services Management Constraints:** Grid Services based on OGSA standards offer interfaces (WS – Resource Properties Document) with a wide variety of information regarding terminology, concepts, operations, WSDL, and XML needed to express the resource properties projection, its association with the Web service interface, and the messages defining the query and update capability against the properties of a WS-Resource. The PbGRMA, through its Service Descriptor component, reaches the corresponding WS-Resource Properties Document per Grid Service to deploy in order to generalize and facilitate the Grid Resource Management process.

These three sources of requirements are important to highlight due to the fact that the PbGRMA is the only architecture that offers this advantage. Current Policy-based Grid

Management Systems [Yan02] [Maga04] only focus on some of these sources of requirements, but our approach is able to merge all of them in only one Domain-Level Grid Service Policy.

The Policy Editor is the entry point of the management architecture. It is the recipient of policies that may have been the result of network operator management decisions or Service Level Agreements (SLAs) between Grid Infrastructure Providers (GIPs) and Grid Services Consumers (GSCs). The SLA requires reservation of resources per Grid Service as well as configuration of the network topology, which is automated by means of policies sent to the Grid-Domain Management System (GDMS). Network-level policies are processed by the GDMS Policy Decision Points (PDPs), which decide when policies can be enforced. When enforced, they are delivered to the GDMS Policy Enforcement Points (PEPs) that map them to element level policies, which are, in turn, sent to the Grid-Node Management Systems (GNMSs). GNMS PDPs perform similar processes at the element level. Finally, the GNMS PEPs execute the enforcement actions at the Grid Infrastructure.

Once the information is introduced, the first task realized by the architecture is to authenticate[1] the user who is accessing the architecture. The authentication task is realized by a special object within the Policy Editor component. The next step is to build an XML policy (Domain-Level Grid Service Policy) using the received information. This task is developed within the Policy Manager component (Figure 3.6). In order to understand the following steps to handle domain-level policies by the PbGRMA, it is important to explain the core policy-based management activities. These activities are related to creating, removing, deploying, and managing every policy instance created by the Policy Manager. In the following section we explain these activities in detail. The use of XML as the format to specify policies in our approach is due to the fact that the Open Grid Forum (OGF) has resolved that communications between Grid components should be based on XML format as per the schemas presented in the following reference [OGF]

3.3.2 Core Policy-based Management Activities

Within the GDMS and GNMS, we identify common activities that are called core management activities (Figure 3.3). The aggregate of these common activities represents another novelty of our policy-based approach. The impact of this architectural conception is that Grid Domain and Grid Node implementations inherit from the core, instantiating its features in order to cope with the specific management functionalities at each level. The following diagram

[1] This thesis is not intended to study concrete security mechanisms; hence, they will only be introduced.

introduces, through a set of use cases, the basic functionalities of these core policy-based management activities. All the functionalities represented by this case are supported by the Policy-based Grid Management Architecture, and therefore by both the Grid Domain and Grid Node Management Systems which extend from it.

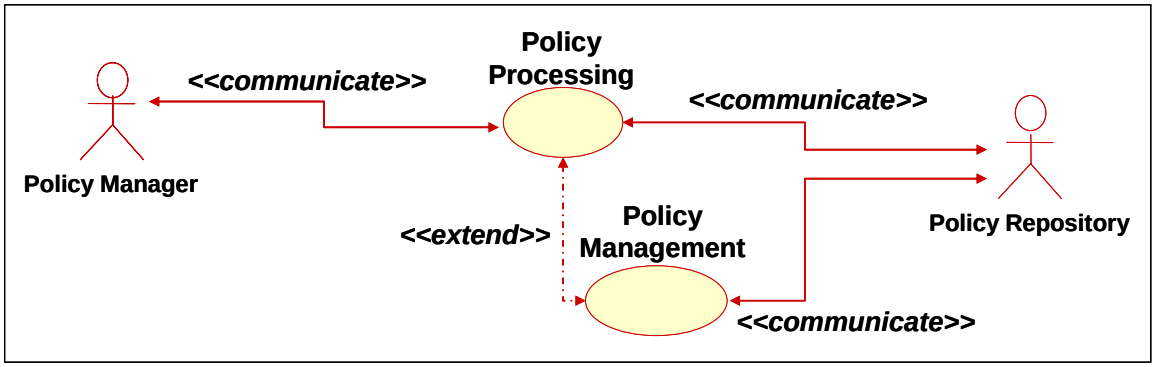

Figure 3.4. Uses case for the Policy-based Grid Resource Management Architecture

Policy Processing: This is probably the most important use case for a Policy-based System. It represents the basic policy processing functionality. That is, the 'provision policy' use case encompasses all functionalities realized in our management architecture each time a policy is introduced in the system. The activity workflow diagram in Figure 3.4 shows the main functionality within the deployment of new Management Instances.

Management Instance is a very important concept in the PbGRMA. When a policy is processed, at least one couple of PDP and PEP is needed to analyze policy conditions (PDPs) and execute the consequent actions (PEPs). The PbGRMA is able to auto-extend basic pairs of PDP-PEPs (e.g. QoS PDP-PEPs and Service PDP-PEPs) in order to simplify the scalability issues in the architecture. When a new couple of PDP-PEP is extended, we call this new extension a Management Instance. In Figure 3.6 below we show QoS and Service PDPs in red boxes, and use green boxes for their corresponding PEPs.

First, the pre-processing functionality, which is realised outside any management instance, checks the identity of the actor that intends to use the management system, through its credentials, and de-multiplexes the policy to the corresponding management instance. A management instance can be seen as a sandbox where all components running have the same owner. Each management instance has, at least one component: the Instances Manager. This component will be described in more detail later on (Section 3.3.3).

Once the policy is dispatched to a particular management instance, the steps that will be followed are as follows:

- Check the authoring actor rights within the management instance. Each management instance has an associated profile. Here, we define what the actor is allowed to do and the maximum amount of resources that can be allocated. This profile has been implemented as an XML schema used to validate the incoming request in the form of XML policies. In this way we claim that our system is secure. GSCs will not be able to request a Grid Service that has not previously been authorized and agreed to between GSC and GIP.

- Extend the management functionality through the download of new components (Management Instances) to correctly process the policy. This feature is explained by the use case titled, 'Deploy Management Functionality.' This activity confers a degree of self-configuration to our PbGRMA.

- Where necessary, extend the management functionality of the PDP by upgrading the action and condition interpreters. The 'Extend PDP Functionality' Use Case further explains this functionality. In this way, our system is flexible and scalable. This property allows the system to be auto-customized. It brings high levels of flexibility and scalability to the entire architecture. Also, for this one and the above mentioned property we claim that the PbGRMA is autonomous.

- Execute the core policy functionality. These are:

    - Store policy in the repository
    - Check policy syntax and semantic conflicts
    - Make decisions about when a policy should be enforced based on events received through the event processing functionality
    - Enforce decisions

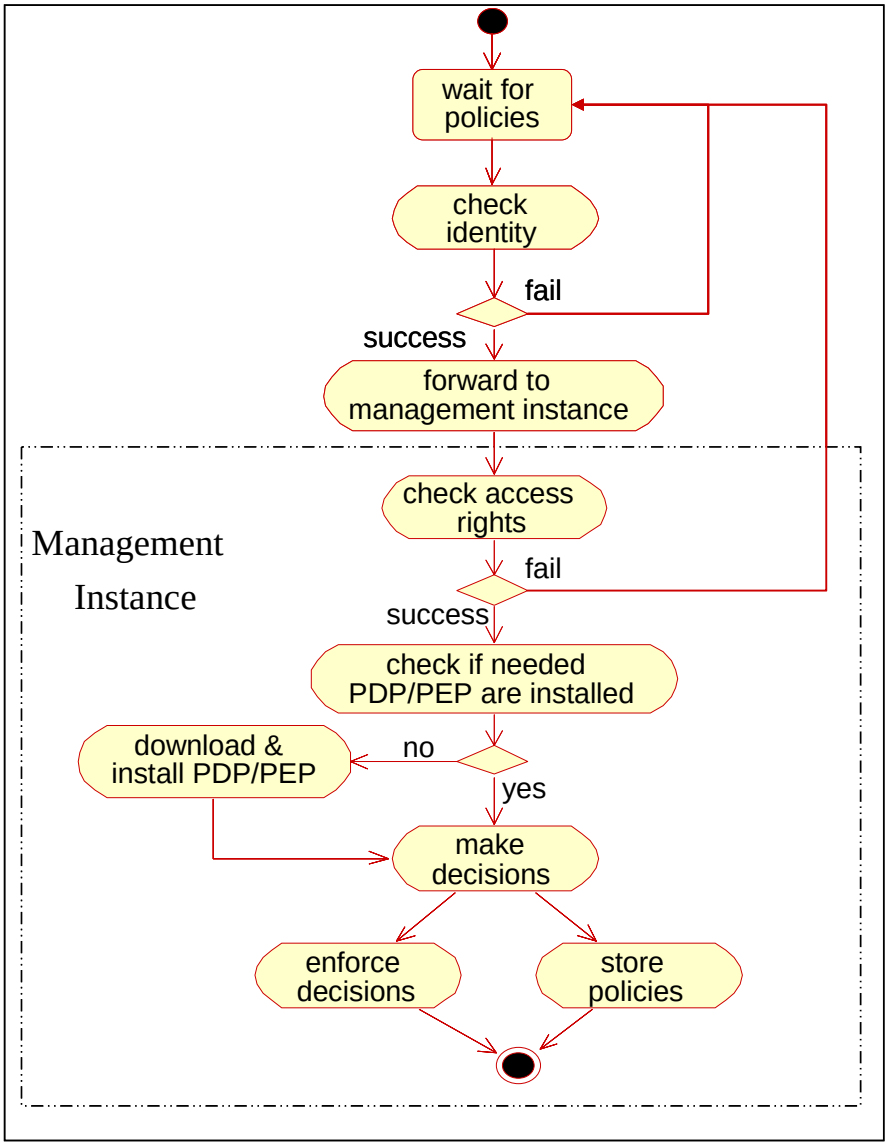

Figure 3.5 Workflow diagram for policy provision and management

Policy Management: As we have explained before, the PbGRMA is able to auto-extend some management capabilities, called Management Instances, when unforeseen management actions are needed at development time. For this reason, it requires an appropriate mechanism to add new functionality at run-time. Once the management system detects that it must be extended, it requests the Repository component (Figure 3.6) to instantiate the required functional domain. Detailed explanation regarding Architecture Extensibility is given in sub-section 3.3.2 of this chapter.

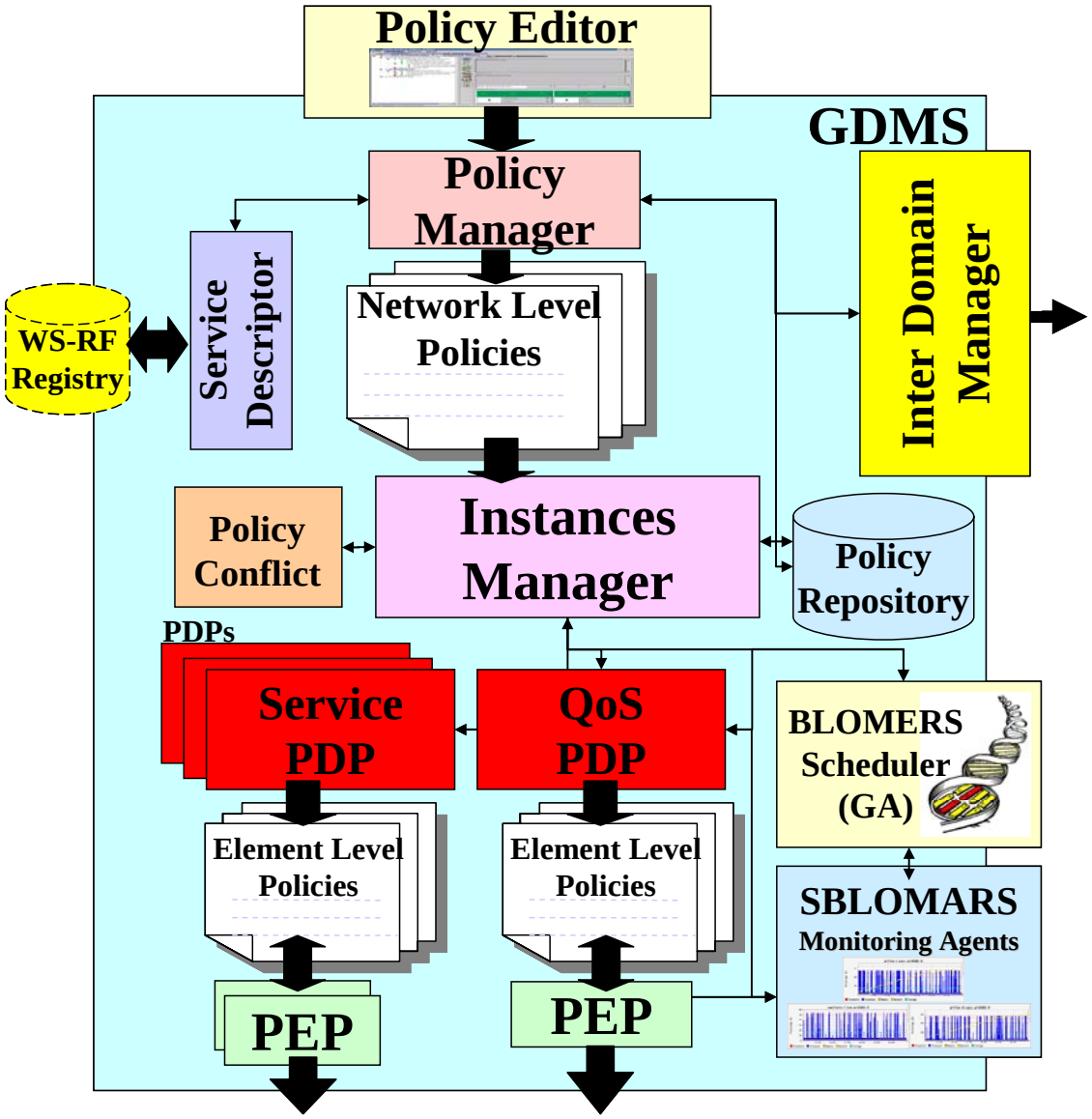

Figure 3.6 Components and interfaces of the Grid Domain-level Management System (GDMS) into the Policy-based Grid Resource Management Architecture (PbGRMA)

3.3.3 Components of the Policy-based Grid Management System

The components of the proposed PbGRMA for Grid Services Management are illustrated in Figure 3.6. They have been developed in order to support service deployment, decision-making with regards to resources control, policy provisions, communication interfaces with WS-Resource Framework, and Inter-Domain Communication respectively. We proceed now to present the details of all of them. As the Grid-Domain Management System (GDMS) and Grid-

Node Management System (GNMS) have similar functionality and components, we focus on the GDMS and, wherever applicable, we note the differences between them.

1. **Policy Editor:** It exists only at the network level. It offers a GUI (Figure 3.7) and a tool-set in the form of templates and wizards for the composition of policies. These are generic enough to accommodate different types of policies, thus exploiting the extension capabilities of the architecture. The policy editor permits creation and modification of management policies in a graphically assisted environment, while supervising the deployment of such policies within the network. To enhance the manipulation of the policy information, the Policy Editor incorporates interpreters enabling the translation of the XML structures into graphical elements that represent each of the policy components. Such graphical elements are then hierarchically arranged in a tree panel, providing the view of the policy layout.

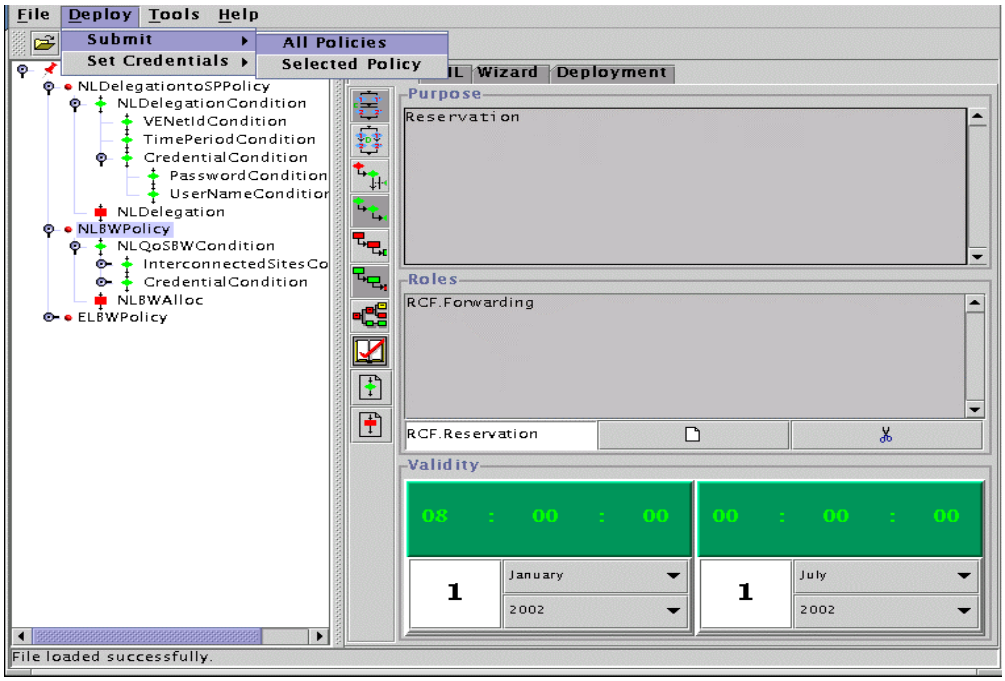

Figure 3.7 The PbGRMA policy editor

In the policy view, it is possible to conduct fine-grain operations: for example, selection of a tree element causes its associated attributes to be displayed in a property sheet so that they can be viewed or changed. In the tree view, new elements may be added directly into the policy structure. This process is easily performed by selecting one of the available policy components (rule, condition, or action) in the toolbar, and subsequently clicking on the desired point in the

tree. Once the policy is considered to be complete, deployment is initiated from the policy editor menu. A validation process is automatically carried out before actually deploying the policy, and the user is informed of any problems. During the policy creation process, the editor gathers and displays reports sent by the different entities involved in the enforcement chain, which allows detection and location of any fault or conflict that may arise, and facilitates its solution.

2. **Policy Manager:** It is a vital component of the architecture. The Policy Manager is in charge of creating Domain-level policies, and basically starting the policy creation process, in order to deploy a required Grid Service. The role of the Policy Manager component is to provide a higher layer abstraction (or adaptation) that offers a global view of Grid resources to the upper layer Grid Services or applications in a more QoS deterministic way. The Policy Manager receives the SLA that has been agreed upon between GIPs and GSCs. This input, together with Grid Service resources and topological requirements received from Service Descriptor Component, is used to start the Grid Resource Management process. Figure (3.8) illustrates the sequence of events explained above.

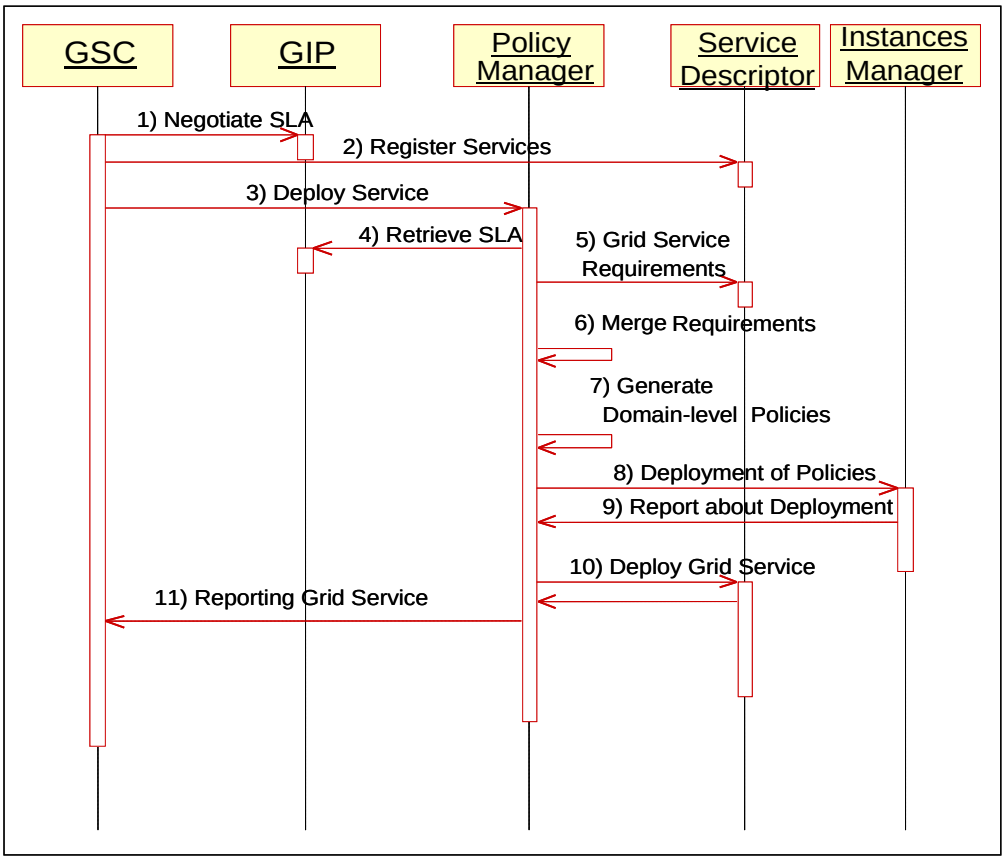

Figure 3.8 The workflow of the creation of a Domain-level policy

In this component the Domain-level Policies are created. There will be only one Domain-level policy per Grid Service requested. We can clearly identify these policies because they simply merge the three source of requirements mentioned at the beginning of this section. These policies have a very high level of abstraction and they need to be refined in detailed policies which are named Node-level Policies. We will describe these policies in detail on sub-section 3.3.4.

3. **Service Descriptor:** The Service Descriptor is compatible with the OGSA standard. It is designed to retrieve the Grid Service Requirements from the Grid Services Factory as well as the Grid Service Instance, and it is one of the most important components in our design. It simplifies communication between the customer of a Grid Service and the Grid Services Factory; normally, the customer must first invoke a service name registry, then request a service instantiation, and finally receive the Grid Service Handler. All these interactions are integrated in this component in order to save communication time, as well as to retrieve the Grid Service Requirements to generate the corresponding service allocation for the client.

This component is designed to retrieve the Grid service resource requirements from the WS-Resource Properties Document. In fact, it integrates a set of processes usually entrusted to the user of a Web Service, namely, to invoke a service name registry, to request a service instantiation, to check whether the WSDL definition of the Web Service Interface is a declaration for a WS-Resource properties document and, finally, to retrieve all the properties elements from this one.

4. **Instances Manager:** It is the core component of the architecture. The Domain Manager receives Domain-level Policies to be dispatched to the appropriate Policy Decision Point (PDP). If the corresponding PDP is not installed, it requests its download and installation. In this way, the management functionality of the system can be dynamically extended at run-time when it is needed. The Instances Manager also controls the lifecycle of these PDPs/PDPs.

The Instances Manager is also contacts the Policy Conflict component in order to ensure that the received policies do not have deployment issues between them. The Domain Manager also acts as a control point, as it has all the necessary information to understand the policy processing state. As an example, imagine that two different policies must be deployed, but the second is only deployed if the first is successfully enforced. In this case, the Domain Manager keeps the second one in a halt state until it receives notification of the first policy's successful enforcement.

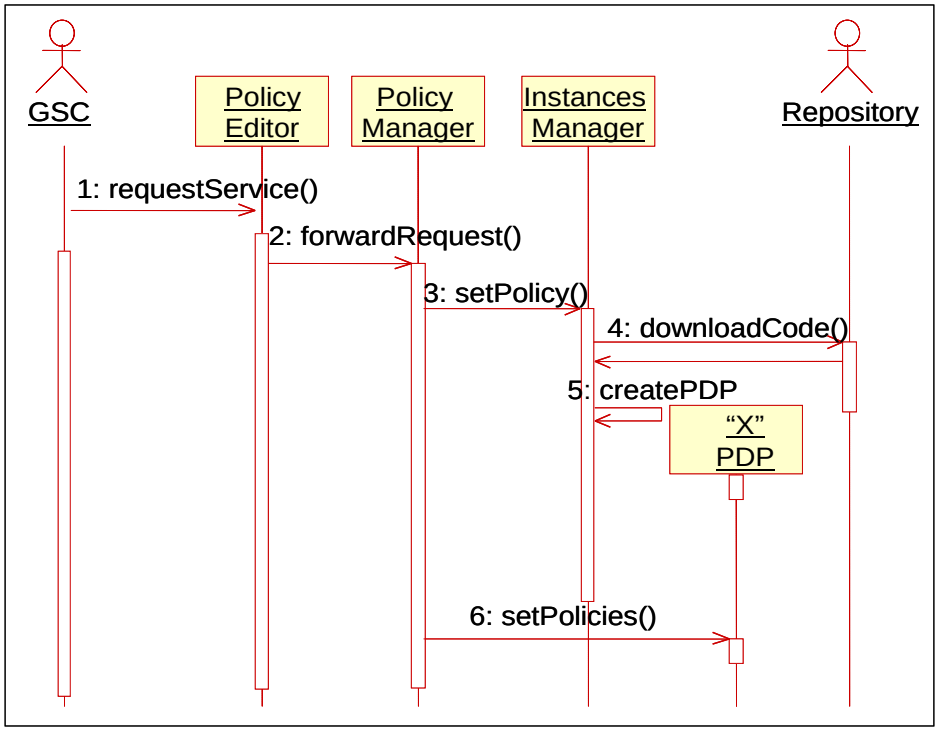

Figure 3.9 Dynamic installation of Policy Decision Points (PDPs)

5. **Policy Conflict Detection:** When policies arrive at specific PDP they should be checked for possible conflicts against other policies previously processed within that PDP. These component is essential because the PbGRMA has multiple PDPs, each covering a particular functional domain (e.g. QoS, Grid Service, etc.).

6. **Policy Decision Point (PDP):** The PDP component checks for possible syntactic and semantic conflicts in policies, solves detected conflicts, makes decisions about when a policy should be enforced, forwards the policies that need to be enforced to PEP components, answers requests for decisions about configuration actions coming from the managed device, and controls the policy validity period in order to uninstall expired policies

7. **Policy Enforcement Point (PEP):** The functionality of the Policy Enforcement Point here is slightly different from that defined within the IETF [IETF]. Here, they receive policies and translate them into the appropriate commands offered by the API of the managed device, for instance in the commands of the Globus Toolkit [Bor05]. In this way the management framework is able to support heterogeneous managed devices, only installing the appropriate PEP component for a particular type of managed device. Since the ratio of management

stations versus managed devices is not necessarily one-to-one but one to many, the PEPs need to distribute the commands to the appropriate managed devices.

8. **BLOMERS Scheduling System:** In Chapter 5 of this thesis we will explain BLOMERS design implementation and features. Here we would only introduce the reader on the role that the scheduler plays in the full Policy-based Grid Management Architecture.

The role of the resource scheduler focuses mainly on assessing resource utilization. This component maintains information about the nodes and links of the system, and can compute possible end-to-end routes for a given service, based on network topology and resource information obtained by the monitoring system. The main role of the RM is to determine a suitable path for the installation of an end-to-end service.

This component is implemented as a stand-alone system. We have been attempting in this research to split the core activities of the Grid Resource Management Process into independent systems. BLOMERS (Balanced Load Multi-Constrained Resource Scheduler) is one of these components. The scheduling process is done by means of the implementation of a heuristic methodology such as Genetic Algorithms (GAs). This approach works in parallel in two ambits: Firstly, it starts different populations (set of resources to match with jobs received) per kind of resource (e.g. memory, storage, processor, etc). Secondly, it starts other populations per level of QoS that has demanded any Grid Service Consumer.

The procedure is triggered by the QoS PDP, which calls the *getSelectedPopulation* operation, providing all the resource and topological requirements of the service. Looking at its internal database, BLOMERS tries to find suitable sets of resources in the Grid, which satisfy the requirements given by the QoS PDP. If the search is not limited by other constraints, e.g. choose shortest path, a set of different paths will result. All of these paths are candidates for the deployment of the jobs that form a Grid Service. The selected paths fulfil the resource and topological requirements of the service. However, a service can have additional requirements, which can be extracted from the service descriptor. BLOMERS does not have access to such information, as this lies within the domain of the ServicePDP, so the ServicePDP makes the final decision. If for some reason the Grid Service cannot be successfully deployed, the QoS PDP can roll back the process by setting the status of this service to “FAILED”, which leads to the removal of all information associated with it. BLOMERS will start any process to offer a new path of resources as candidates to execute the corresponding job.

9. SBLOMARS Monitoring System: In chapter 4 of this thesis we will explain SBLOMARS design implementation and features. Here we only wish to introduce the reader in the role that the scheduler plays in the full Policy-based Grid Management Architecture.

The SNMP-based Balanced Load Monitoring Agents for Resource Scheduling in Grids (SBLOMARS) are logically and physically distributed in the overall Grid Infrastructure (i.e. Grid Resources). The monitoring system effectively collects analyses and provides the necessary information needed by the BLOMERS resource scheduler to make appropriate decisions. This distributed monitoring system involves a set of autonomous and distributed monitoring agents, which generate real-time and statistical availability information for every resource and entity composing the Grid.

SBLOMARS is able to monitor processor, storage and memory use, network activity at the interface level, services available (an updated list of applications installed), and end-to-end network traffic. This can be done across different architectures, including Solaris, Unix-based, Microsoft-based, and even Macintosh platforms. Moreover, SBLOMERS is self-extensible to monitor multi-processor platforms to huge storage units. Specifically, it is designed around the Simple Network Management Protocol (SNMP) to tackle the generality and heterogeneity problem, and is also based on autonomous distributed agents to facilitate scalability in large-scale Grids.

SBLOMARS helps solving the scalability problem by the distribution of the monitoring system into a set of sub-monitoring instances which are specific to each kind of computational resource monitored. Therefore, it deploys a single software thread per type of resource to be monitored, independently of the amount of such resources. This is worthy of mention because many monitoring systems fail when they try to handle new resources that have been added to the system. As we mentioned before, every resource is monitored by independent software threads that start again after a certain amount of time, becoming an infinite cycle. The cycle-timing is defined by local or remote administrators through booting parameters at the beginning of its execution. SBLOMARS automatically re-configures their polling times (calls to SNMP daemon) in an autonomous way. SBLOMARS increases or decreases the interval times between consecutive interrogations to the same resource based on the state of the monitored devices. This feature is important where the overload caused by any monitoring system is not affordable by the hosting nodes.

SBLOMARS also introduces the concept of dynamic software structures, which are used to monitor from simple personal computers to complex multiprocessor systems or clusters, even with multiple hard disk partitions. These features make our approach novel compared with similar monitoring systems, such as Ganglia [Mas04] that is restricted to general resources and

not when they are fragmented. Another novelty in our monitoring system is that SBLOMARS integrates an end-to-end network-level monitoring technology. It is the CISCO IOS® IP Service Level Agreements, which allows users to monitor end-to-end network-level performance between switches, routers, or from either remote IP network device.

10. Inter-Domain Manager (IDM): The IDM is in charge of implementing end-to-end negotiation of service deployment into separate Grid nodes that belong to different administrative domains, managed by different organizations. Each Grid Node Management System (GNMS) only manages a single Grid Node. A Grid Domain Management System (GDMS) oversees these GNMSs, and the IDM is located within the GDMS. We adopt the Java Remote Method Invocation (RMI) as the communication channel for the distributed system.

In our design, we established a repository that maps a destination address to an IDM, i.e., a 'many-to-one' relationship. As such, a GDMS must register a list of destination addresses within its domain on a repository that has a well-known address, so that this repository can provide a discovery mechanism for mapping destination addresses to their respective IDM. Within a GDMS, the IDM interfaces with two other key components, i.e., the Policy Manager and the Instances Manager.

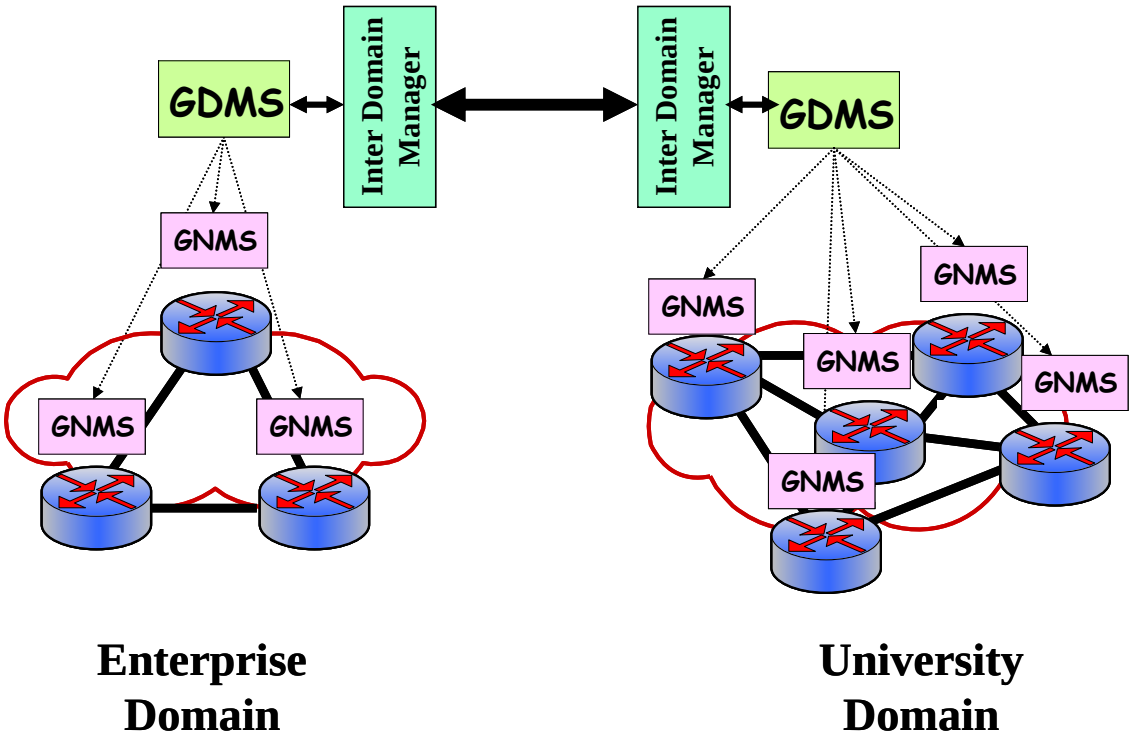

Figure 3.10 Simplest form of Inter-Domain Interaction

The effect of an inter-domain view can be described with the example represented in Figure 3.10, where we have two domains called Enterprise and University. The BLOMERS Grid Resource Manager will invoke the *relay (Context, ServiceDescriptor)* method on the IDM in the University Domain .*Context* represents the ingress IP and the destination IP of the reserved route between both domains.. For *ServiceDescriptor*, it would be sufficient to input the service component name. The Resource Manager should have no problem extracting this information from the Grid Domain-level Policy information that it has. The intra-domain process is stalled at this point. At the other end, the IDM in Enterprise Domain will request the IDM in University Domain to extend the Resources Activation in the neighbouring domain. The *deployService (ServiceName, VANNode[], Credential)* method of the Policy Manager component will be invoked by the IDM in University Domain. The VANNode array is a pair of ingress-destination IP address abstracted from the context information obtained from the Policy Manager. The IDL description for the GDMS interface is shown in Figure 3.11.

```
#ifndef _idm_IDL_
#define _idm_IDL_
module nms {
   module idm {
      typedef string IPAdd;
      typedef string QOSParameters;
      struct Context {Domain-level QoS policies
         IPAdd ingress;
         IPAdd destination;
            };
            struct QoS {
                  QOSParameters bandwidth;
                  QOSParameters cpu;
                  QOSParameters memory;
            };
            // Provides method for inter-domain reservation
            interface Requestor {
                  exception InvalidRequest {};
                  void reserve((in
org::ist_pbgrm::network::netasp
                    ::ServiceDescriptor descriptor,
                    n Context c)) raises ((InvalidRequest));
               String trade((in QoS q)) raises
((InvalidRequest));
            };
            /** Gets the correct IP address of the NMS and port
                  interface Repository {
                  exception MappingNotFound {};
     IPAdd getLocation((in IPadd destination))
     raises ((MappingNotFound));
                  void setLocation((in IPadd destination));
            };
      };
```

Figure 3.11 IDL Description for the Inter-Domain Communication Process

11. Policy Enforcement Point (PEP): It receives policies and translates them into the appropriate commands offered by the API of the managed device, for instance in commands of the Globus Toolkit. In this way the management framework is able to support heterogeneous managed devices, only installing the appropriate PEP component for a particular type of managed device.

12. Policy Repository: The primary role of the database is to meet the management framework software component's needs. Typical information stored in the database could be policies, domain components, profiles, access rights, topological information, etc. The policy repository is supported on a LDAP directory, there is only one LDAP directory per Grid Domain Management System, which provides content-based policy searches and distribution transparency. These features make it suitable for providing scalable storage solutions in network-wide systems, such as the Fain management system [Sal03].

In general, accessing LDAP directories from Java applications is enabled through the use of the JNDI (Java Naming and Directory Interface) API. JNDI offers an abstract view of the LDAP directory, hiding the LDAP specific operations under a standardized interface suitable for interacting with different storage services that follow a similar approach. The following diagram illustrates the basic building blocks of the repository.

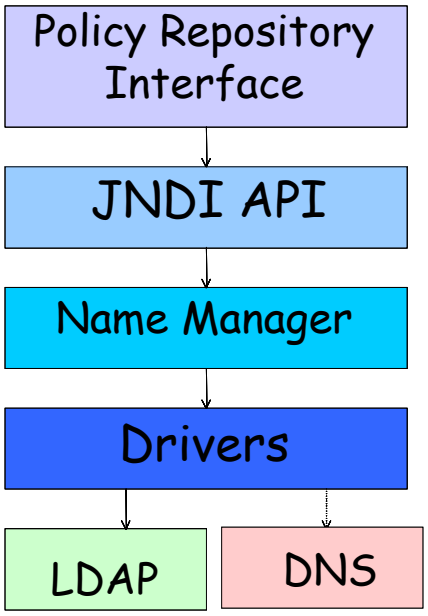

Figure 3.12 Policy repository access interface structure

The LDAP directory might be accessed directly from the PDP internal components or through an intermediate cache that would improve the overall efficiency.

- A reduced set of requirements have determined the policy repository design, namely: hard and soft constraints.
- Access to the repository shall be technology transparent. Neither LDAP nor JNDI specific issues shall be exposed to the outside components visiting the policy database.
- A centralized component (controller) shall organize the directory look-ups.
- The policy directory shall provide searching mechanisms based on policy, condition or action-specific attributes.

As a design decision to enhance performance, only essential look-up attributes are stored in the directory. Since only java applications are intended to access the directory, the impact of such decisions on interoperability is limited.

The policy directory basically consists of a database access controller and a series of state factories and object factories appropriate for the different policy classes. The database access controller provides a simple query interface for efficient database searching operations. The main responsibility of the controller is to locate the requested policies, retrieve them, and return them in an appropriate format. The controller is also in charge of directing the storage process according to the specified hierarchical relationships.

The state factories and objects factories are merely format translators that convert java objects into LDAP entries and vice versa. Each class that may be stored in the repository must implement the Storable Interface that contains operations for providing hierarchical information not contained in the schema, and its identifier (a distinguished name that must be obtained, so that there is no collision when storing the entry in the directory).

The policy schema defines the type of policy classes that can be stored in the LDAP database and the valid attributes for each of them. The hierarchical relationships existing between the classes is not reflected in the schema but maintained on the directory structure itself (as in the case of a file-system).

3.3.4 Flexibility and Extensibility of the Architecture

One of the most important features in the PbGRMA is its ability to self-extend its management capabilities in order to handle new functional activities, which are identified in our context as Management Instances. We would remember that a management capability is the ability of the PbGRMA to manage a set of actions that are related to the same target.

In the PbGRMA we have implemented two management capabilities, which are, “Service Deployment” and “Quality of Service.” Another management capability could be the ability to offer, “Fault Tolerance,” in the deployed services. In our approach, these management capabilities are known as Functional Domains.

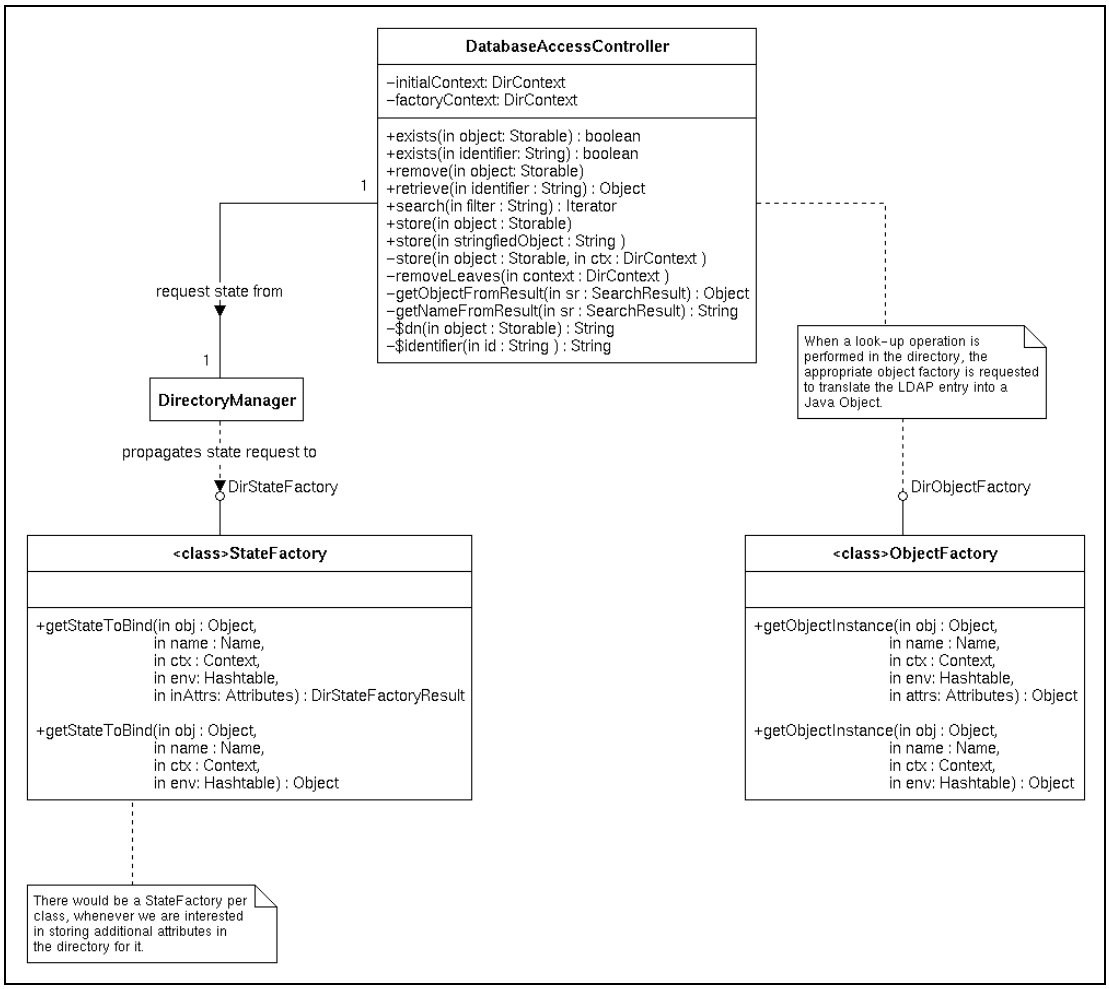

Figure 3.13 Policy repository class diagram

The ability to offer these management capabilities requires implementation of several mechanisms and management actions. These actions are known as functional activities. Therefore, a management capability involves a set of functional activities. In our approach, the mechanisms to extend new management capabilities are implemented, and we will describe this process in the following paragraphs. Fault Tolerance is not yet implemented in the PbGRMA but its integration is quite simple, due to the fact that the architecture offers the necessary mechanisms to extend it from the Domain Manager component.

When a functional domain is already deployed (QoS PDP or Service PDP), not all the functional activities are guaranteed to be available. These functional activities rely on specific Conditions Interpreters that are produced when an Action is accepted. We have reduced the number of functional activities alive in the first deployment of the management architecture in order to reduce the overhead of the management architecture. This is to save the maximum amount of resources needed to continue executing the PbGRMA.

We will clarify this feature by means of an example. A Grid Service Costumer is requesting a, "Video Decoding Service." The user contacts the Policy Editor component to deploy the specific service policy. Once the processes taking place in the PbGRMA are completed (collaboration between SBLOMARS, BLOMERS and Domain Manager), the service is ready to be deployed along the Grid Infrastructure, through the Grid Node Management System PEPs. So far, only the Service PDP (functional domain) has been used by the architecture to deploy the requested service because there are no QoS policies requested. When a QoS policy is implicated, the architecture needs to extend a new functional domain, incorporating the QoS PDP. Although the new domain has been incorporated, some activities inside this domain are not yet available. The PbGRMA follows the same philosophy, because it only deploys the instances which are required to execute all management policies.

Therefore, the PbGRMA is dynamically extensible at two distinct granularity levels, the PDP-PEP (Functional Domain Level) and the Action-Condition Interpreters (Functional Activities Level). In both cases, the extension might be triggered during the policy processing if the system requires a component not yet installed. In the following paragraphs we will describe both extensions of the PbGRMA.

I. Extending Management Functional Domains: It is the first level of extensibility in this approach. When a new management domain is required, and it is not already installed on the system, a new functional management domain deployment process is triggered automatically by the PbGRMA. This action is one of the main activities of the Domain Manager component. The first instance is the Domain Manager, which is responsible for forwarding received policies to the appropriate Domain Policy Decision Point (PDP). If the corresponding PDP is not installed, the Domain Manager requests the Repository to download and install it, thereby extending the management functionality of the system as required.

The sequence diagram in Figure 3.14 illustrates how the aforementioned extension is achieved. Then a request is raised to the Repository that initiates the installation of that component. Once the PDP is ready, the Domain Manager forwards the policy to it normally. Sometimes it is possible to deploy more than only one Domain PDP but it will depend of the

requirements into the Domain Policy that is trying to be executed. Often, the needed functional extension would not require the introduction of a complete management service. but just the extension and/or modification of one already available. It means than some components of the already installed Doamin PDPD could be added or removed.

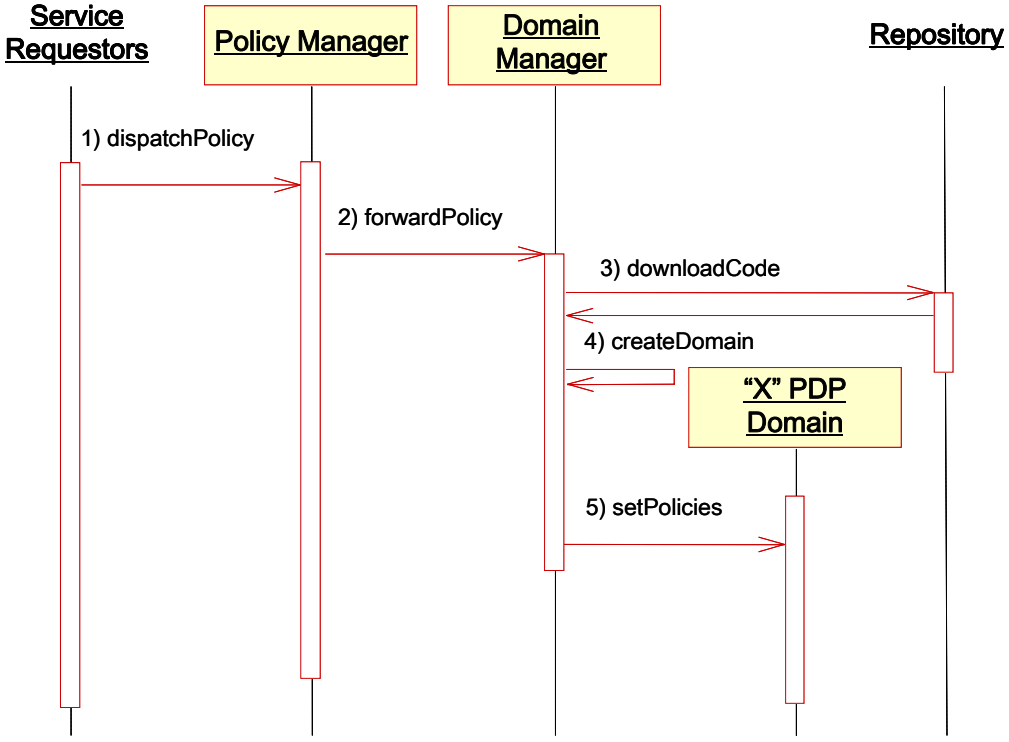

Figure 3.14 Extending management domains

II. Extending Management Functional Activities (Action-Condition Interpreters): The dynamic installation of new Action and Condition Interpreters is another option for extending management functionality, in addition to the installation of PDPs previously described. Two key classes within the PDP component are the action and condition interpreters. They provide action and condition processing logic for groups of policies handled by the PDP. Each PDP has at least one action and condition interpreter, although they might have more. Within each PDP, although drawn separately, there is a generic Action Interpreter class and a Class Loader class. The generic Action Interpreter class receives all requests to Action Interpreters and de-multiplexes them. When a requested Action Interpreter object is not found, it interacts with the Repository to download the needed code. Figure 3.15 shows the interactions occurring when this happens.

III. Removing Management Domains and Interpreters: Any new deployed instance has a limited lifetime. It is a counter that is decremented once the component has finished its corresponding management activities. Once the deadline is reached the component is removed from the system. Therefore, this facility contributes to the system autonomy.

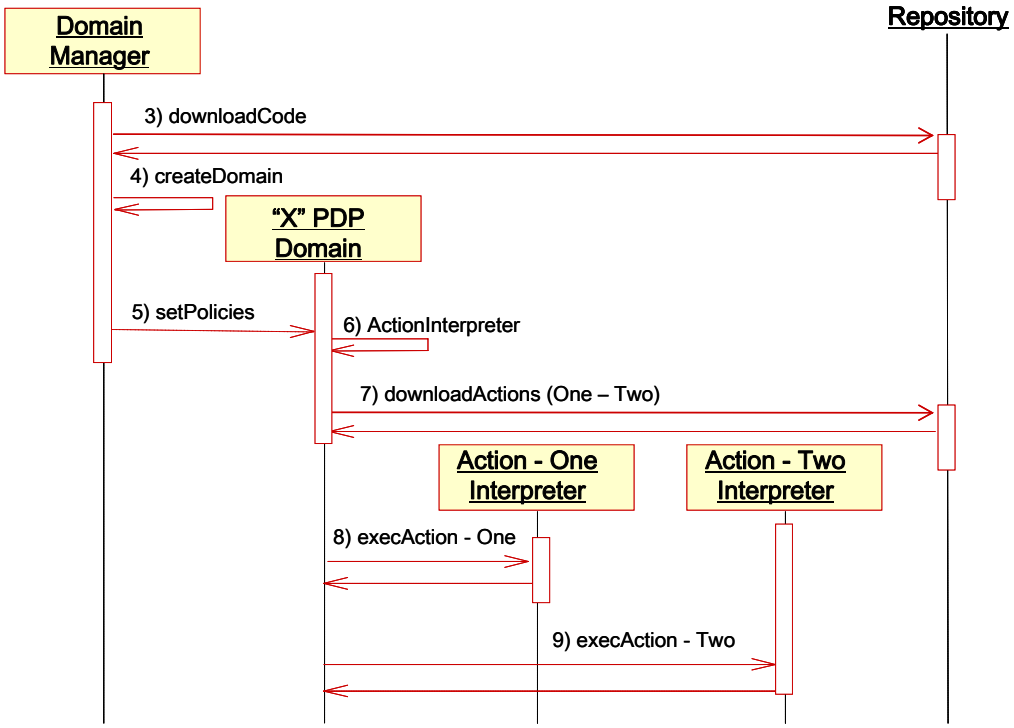

Figure 3.15 Extending management actions – Conditions interpreters

3.3.5 Structure of the Management Policies

Domain level policies and node level policies are expressed in XML (eXtensible Markup Language) [XML] and transmitted using SOAP (Simple Object Access Protocol). SOAP permits the transmission of policies as plain XML, ensuring interoperability.

Data-types are supported, and there is the ability to specify relationships and constraints between different elements of a document. The architecture presented takes advantage of the properties of the XML Schema to reflect the access rights of users in relation to management functionality. That is, the framework is capable of dynamically creating, as a result of management policy enforcement, restricted XML Schemas for particular users against which XML user policies are validated. When a user obtains certain management

responsibilities from the network operator, by means of delegation, it is effectively assigned one or more restricted XML Schemas, which delimit the types of policies and the policy action and condition values allowed to that user. The policy structure used in our approach is based on the IETF Policy Core Information Model [IETF] though simplified by defining as mandatory only those features essential for policy processing. Hence, the size of policies is considerably reduced (around five times smaller than following the PCIM model) and their processing is simpler.

A section of the policy schema used in our evaluation tests is depicted in Figure 3.16. The policy rule consists of seven elements. First, the ***PolicyRuleName*** uniquely identifies the policy within the management infrastructure. Hence, in addition to the policy type identifier, a sequence number is included.

```
<xsd:schema targetNamespace=http://nmg.upc.edu/~emagana/pbma_Schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http:// nmg.upc.edu/~emagana/pb_Schema ">
<xsd:element name="PolicyRule">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="PolicyRuleName" type="format_Type"/>
            <xsd:element name="PolicyRoles" type= "… "/>
            <xsd:element name="UserInfo" type= "… "/>
            <xsd:element name="PolicyRuleValidityPeriod" type= "… "/>
            <xsd:element name="PolicyDomain" type= "… " minOccurs="1"/>
            <xsd:element name="Conditions" type= "… " minOccurs="1" maxOccurs="unbounded"/>
            <xsd:element name="Actions" type= "… " minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 3.16 Policy schema example

The ***PolicyRoles*** element identifies the roles to which the policy applies. That is, all network elements developing a role listed in the policy are expected to respond to it.

UserInfo contains the identifier of the user that is introducing the policy into the framework. This identifier is used to select the restricted XML Schema against which the user policy should be validated.

The policy expiration date is contained within the ***PolicyRuleValidityPeriod*** element. The expiration date is usually given with the day and hour the policy starts and finishes being valid. Nevertheless, filters specifying concrete months, days, and hours during which the policy is not valid can be also introduced.

The ***PolicyDomain*** element is used for the correct processing of policy sets. A policy set is a group of policies that should be processed in a particular way, i.e. atomically, sequentially, etc.

The ***Conditions*** element includes all policy conditions. Conditions can be either compound or simple and refer to an hour of the day, an IP flow, a concrete notification, or a managed device status. The modules needed to monitor these conditions, if any, are also extracted from the Conditions element information. This element is optional. When not included, the framework interprets that the policy action should be enforced directly.

The ***Actions*** element contains the action type and parameters, as well as information about the module responsible for enforcing this action. At least one Actions element is mandatory in all policies, but there can be more than one. The defined policies have been categorized according to the domain of management operations, policies that belong to a specific domain are processed by dedicated Policy Decision Points PDPs and PEPs.

3.4 Overall view of the Grid Resource Management Process

The Grid Service management process starts when an authorized user requests a Grid Service to the PbGRMA through the Policy Editor Interface. At this time the client will specify the name of the requested Grid Service and the QoS requirements for its deployment, if they are required. Once the PbGRMA receives the client request, it exchanges information related to the service in order to process the resources service requirements (Figure 3.17).

I. Service Level Agreement (SLA): Previous to any Grid Service request, every GSC has concluded a SLA with at least one GIP. In this agreement a specific QoS level will be satisfied for every service running on their provider's infrastructure. We assume that QoS is quantified in levels like, "diamond," "gold," "silver," and "bronze".

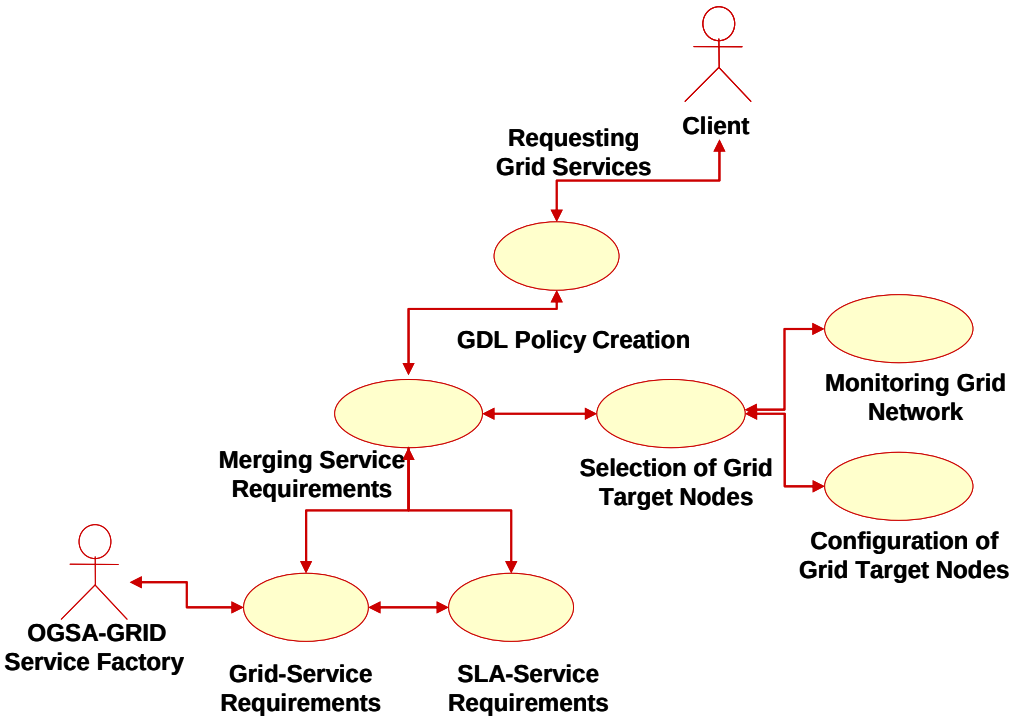

Figure 3.17 Use case for deployment and management of Grid Services

II. Grid Services Management (GSM) Phase: Our architecture merges requirements from the client, the provider's resources availability, as well as the service specification requirements, and creates the Grid Domain Level (GDL) Policy for the corresponding Grid Service. This feature is one of the most important novelties in this approach. In the context of this work, the

Policy Manager must contact the Service Descriptor in order to receive the Grid service specifications, which will be extracted from WS-Resource properties document. This document has been associated with the WSDL porType defined by Grid Service Instance requested via UDDI Registry. In Figure 3.18 we show fragments of the above-mentioned documents, which were used during the evaluation phase described on Chapter 6. The process of building the Domain-level policy also involves the integration of two other features. The OGSA compatibility and Inter Domain communication. We will describe these communication processes in the following two paragraphs.

A. **OGSA Compatibility:** As previously stated, the proposed architecture is compatible with OGSA. The Service Descriptor Component uses XML requests to tag service data, SOAP to transfer it through the network, WSDL for describing the services available, and finally UDDI is used for listing what services are available. SOAP, as a data communication format, offers different advantages in order to extract the information necessary for our architecture. Essentially, we parse the SOAP files into the OGSA Policy Descriptor and extract the Grid Service specifications mentioned above. The Service Descriptor sends back the parsed information in Java format to be processed by the Policy Manager and to create the GDL Policy.

```
<!-- ========== WSDL Interface for Newton's Method Application  ============ -->
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://nmg.upc.es/Newton'sMethodExample" ...>
 <wsdl:types>
    <xsd:schema
      <xsd: import
    targetNamespace="http://nmg.upc.es/NewtonsMethodExample_Properties"
      <xsd:attribute name="ResourceProperties" type="xsd:Newton's Method"/>
      ...
<!-- == WS-Resource Properties Document for Newton's Method Application  == -->
<wsdl:portType name="Newton's Method"
    wsrp:ResourceProperties= "intf:GenericMethodProperties">
    <xsd:sequence>
       <xsd:element maxDistribution="5" minDistribution="1" name=" " …
       <xsd:element amountMinMemory="20" amountMaxMemory="250" name=" "...
       … "wsa:EndpointReferenceType"/>
</wsdl:portType>
```

Figure 3.18 WSDL and WS-RF interfaces.

B. **Inter-Domain Communication:** Just in the case the requested service needs resources that belong to different administrative domains, the Policy Manager contacts the Inter Domain Manager to start the resources negotiation with the other domain. Due to the fact that communications are always under XML format, there should not be problems regarding communication amongst management entities. At this time, the second domain is starting its own resource management and scheduling processes.

III. Grid Resource Discovery and Monitoring (GRDM) Phase: The Policy Manager (PM) sends the just created GDL Policy to the Domain Manager. It later evaluates the conditions of the policy and tries to match them with the available resources in the Grid Infrastructure of its local domain. In order to complete these functions, it first has to analyze the QoS of the requested service. To do so, it contacts the BLOMERS Resource Scheduler, whose task is to analyze the network topology and resources information received from the SBLOMERS Monitoring System. The checking of available resources is realized for every resource involved in the establishment of a service into the Grid topology (i.e. memory, bandwidth, storage, etc).

IV. Grid Resource Scheduling (GRS) Phase: BLOMERS tries to find a suitable set of Grid nodes that satisfies the requirements given by the GDL Policy. If the search is not limited by other constraints, a set of different nodes will result. All these Grid nodes are candidates for the allocation of the service because they fulfil the resource and topological requirements expressed in the GDL Policy. However, a service has additional requirements specified in the Grid Service Data. BLOMERS does not have access to such information because this lies within the domain of the Service PDP. Therefore, the Domain Manager decides the set of final nodes to allocate the service.

V. Grid Nodes Configuration – Grid Node Level Policy Creation: At this time, the Domain Manager has to decide which resources will be part of the final set of Grid nodes that will be configured to execute the Grid service with the specified QoS level. In this component, an appropriate algorithm carries out the selection of the best nodes and forwards the policy to the QoS PDP. Next, the QoS PDP sends the decision to its corresponding PEP. The QoS PEP will transform the request into a set of appropriate Grid Node Level QoS Policies (one policy for each of the Grid nodes selected) and it will send the policy into the GNMS of the established nodes. Once the QoS policy is enforced, the GNMS calls an activation method of the local node interfaces for each node (i.e. Globus Toolkit Primitives), thus ending the configuration process.

VI. Jobs Allocation and Activation (JAA) Phase: Since the enforcement of the QoS policies has successfully terminated in all nodes, the Domain Manager starts processing the activation policies by forwarding them to the corresponding PDP, to be evaluated. If there are no conditions or actions to be processed at the domain level, it forwards the policy to the PEP of the involved Grid Nodes. The activation PEP enforces the policy that assigns the resources using the interfaces offered by local node interfaces at each node. The result of the Grid resources activation is forwarded back to the Grid Domain Level through the Grid Node Level for control and fault management purposes. At this time, the service requested is running with the agreed QoS level. The SBLOMARS Monitoring System updates the QoS PDP with resource utilization. Therefore, in case one or more of the Grid Resources can not offer part or the totality of their committed capacity, the PbGRMA will restart the Grid Service Management Process to find new Grid Resources which offer similar capabilities.

3.5 CONCLUSIONS

In this chapter we have described how our approach performs the complete Grid Resource Management (GRM) process. The Policy-based Grid Resource Management Architecture (PbGRMA) presented in this chapter is mainly in charge of solving the first and last phases in the Grid Resource Management process, which are the Grid Resource Management (GRM) and the Jobs Allocation and Activation (JAA) phases. Therefore, we start and close the loop for Grid Services deployment and management by means of an architecture based on policies.

The presented PbGRMA is integrating three sources of Grid Services requirements: users' requirements in QoS terms, Grid Services specifications based on the OGSA standard and Grid resources availability to maintain certain load balance along the Grid Infrastructure.

In this chapter we have explained the features, implementation details, and advantages of the PbGRMA. In particular, we described how our approach is able to deploy and to manage Grid Services instances into heterogeneous networks configured such as Grid Infrastructure. The PbGRMA extracts the service requirements from different sources and collects information about resource availability in the Grid Infrastructure Providers domain through the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS). Finally, it schedules Grid Services on best available computational resources by means of the Balanced Load Multi-Constrained Resource Scheduler (BLOMERS), getting load balancing through all of the nodes.

Although our approach is focused on Grid environments, where the wide range of nodes will offer small amounts of resources, the solution is not limited to this domain of users/clients. The whole architecture is completely scalable and flexible in terms of the type of elements to be managed. Also, it is Web Services Resource-Framework oriented. This feature guarantees that our approach follows current Open Grid Forum (OGF) standards.

One of the most innovative aspects in our solution is its ability to auto-extend many of its management capabilities. The PbGRMA auto extends Domain-level and Node-level components such as Policy Decisions Points and Policy Enforcements Points. The PDP components are also self-extended in Actions and Conditions Interpreters. This solution offers a great level or granularity, thus helping to scale properly. Moreover, the auto-configuration of the architecture to deploy or remove new Management Instances improves the resource used by the management system. For all these properties we claim that our PbGRMA is an autonomous management system.

On the side of drawbacks, The PBGRMA lacks of simplicity. This system is not very easy to deploy without the proper knowledge of their components and the structure of the system. The interfaces with SBLOMARS and BLOMERS are based on XML standards. It means that

transformation of the information from XLM-based documents to policy language and even software language is a very hard activity. In fact, such transformations could take long time. Unfortunately, this is the price to pay for keeping the solution with high levels of standardization.

Chapter 4

RESOURCE DISCOVERY AND MONITORING IN LARGE-SCALE GRIDS

4.1 INTRODUCTION

Resource discovery and monitoring is defined as the process of dynamic collection, interpretation, and presentation of information about hardware and software systems. It is one of the crucial activities in the Grid Resource Management (GRM) process. There are several kinds of discovering and monitoring: monitoring of resources (availability), network monitoring (connectivity), and jobs monitoring (status of jobs). In this chapter we will describe our research in resource and network monitoring. Jobs monitoring is outside of the scope of this thesis.

As we described in the first chapter, Grid Computing started as a technology to offer high-performance computing to external users with high-throughput applications needs. Up to the present, this technology has received exceptional acceptance from academic and commercial areas, becoming a focus of research in many institutes, universities, and enterprises. Over time the Grid has modified its initial objectives to become a technology where non-massive resource owners would be able to share their resources and integrate human collaboration across multiple domains, regardless of network technology, operative platform, or administrative domain. This novel technology is known as Large-scale Distributed Computational Grids.

This evolution was adopted by all research areas inside GRM. The area of GRDM was one of them. Eventually, the ability to monitor distributed computing components becomes critical for enabling high performance computing in a large complexly distributed environment, such as Grid Computing. However, achieving performance in a Grid environment is often difficult. The Grid itself does not provide a stable platform. Its environments are dynamic, heterogeneous, and subject to frequent faults. Timely and accurate dynamic performance information is required for a variety of tasks, such as: performance analysis, performance tuning, performance prediction, scheduling, fault detection and diagnosis, etc.

The aim of this chapter is to describe the contribution of the thesis to resource discovering and monitoring in Grid Computing. The material incorporated herein describes the design and functionality of the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling in Large-scale Grids (SBLOMARS). The open distributed monitoring system covers the GRDM phase of the Grid Resource Management process proposed in this thesis.

The structure of this chapter is as follows: After this Introduction, in Section 4.2, we briefly describe the Grid Resource Discovering and Monitoring process and the current standards accepted by the Open Grid Forum (OGF). Section 4.3 is an overview of our SBLOMARS. In this section we will describe the design, implementation, functionality, and main features of our Grid monitoring system. In Section 4.5 we present the conclusions of this chapter.

4.2 GRID RESOURCE MONITORING AND DISCOVERING

We have explained that discovering and monitoring of network and computational resources in Grid Computing are not simple activities. Actually, the Open Grid Forum (OGF) [OGF], which is the new organization that resulted from the merger of the Global Grid Forum (GGF) and the Enterprise Grid Alliance (EGA), is working in autonomous specific groups to define standard mechanisms to improve and generalize resource discovering and monitoring in Grid Computing. In their last draft [Tie03] they have detailed features and recommendations for Grid Monitoring[1] systems or architectures. Below is a summary the most important of them:

I. **Minimizing Latency:** As previously stated, performance data is typically relevant for only a short period of time. Therefore, it must be transmitted from where it is measured to where it is needed, with low latency.

II. **Handling High Data Rates:** Performance data can be generated at high data rates. The performance monitoring system should be able to handle such operating conditions.

III. **Reducing Overload:** If measurements are taken often, the measurement itself must not be intrusive. Further, there must be a means of monitoring facilities to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.

IV. **Increasing Security:** Typical user actions will include queries to the directory service concerning event data availability, subscriptions for event data, and requests to instantiate new event monitors, or to adjust collection parameters on existing monitors. The data gathered by the system may itself have access restrictions placed upon it by the owners of the monitors. The monitoring system must be able to ensure its own integrity and to preserve access control policies imposed by the ultimate owners of the data.

[1] Discovering and Monitoring will be referred to as monitoring from this point onward.

V. **Being Scalable:** This is one of the crucial requirements of Grid Monitoring and Discovering. Because there are potentially thousands of resources, services, and applications to monitor, and thousands of potential entities that would like to receive this information, it is important that a performance monitoring system provide scalable measurement, transmission of information, and security.

In order to meet these requirements, a monitoring system must have precise local control of the overhead and latency associated with gathering and delivering the, "resources availability information"*.* We understand resources availability information in the context of this thesis as the information reported by a monitoring entity about the state of specific resources, such as amount of memory, CPU cycles per second used, and storage capacity. In the Grid, the amount of resources availability information will be directly proportional to the amount of resources available on the Grid. Typically, there is a huge amount of resources in the Grid, and searches of this space will have unpredictable latencies. These potentially large latencies must not impact every request for performance information. Instead, searches should be used only to locate an appropriate information source or sink, whereas operations with a more predictable latency should be used to transfer the actual performance information.

The proposed core Grid Monitoring Architecture from the OGF consists of three main components, as shown in Figure 4.1. Grid monitoring architecture is designed to handle performance data transmitted as time-stamped (performance) events. An event is a typed collection of data with a specific structure that is defined by an event schema. Performance event data is always sent directly from a producer to a consumer. The Producer makes performance data available (performance event source). The Consumer receives performance data (performance event sink). The importance of this architecture is that the Grid monitoring system presented in this chapter is grounded on it..

Directory Service supports information publication and discovery. Producers (and consumers that accept control messages) publish their existence in directory service entries. The Directory Service, or Registry, is used to locate producers and consumers. Note that the term, "Directory Service," is not meant to imply a hierarchical service such as LDAP [LDAP07]; any lookup service could be used. The directory service serves to bootstrap communication between consumers and producers, as entries are populated with information about understood wire protocols and security mechanisms.

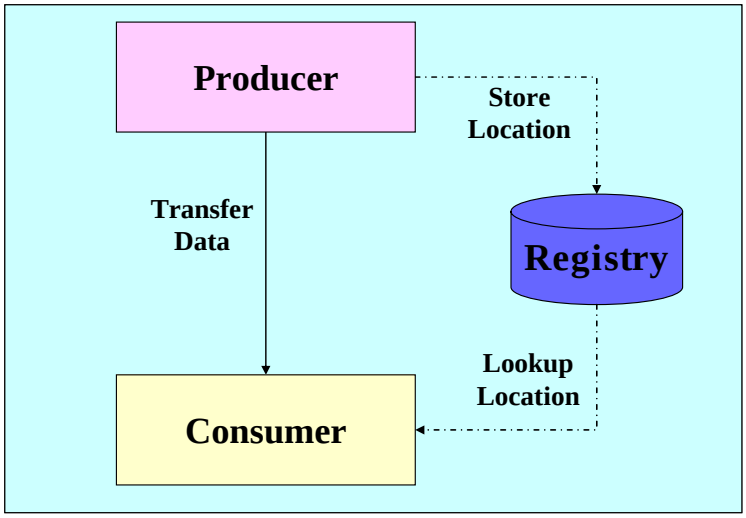

Figure 4.1 Grid Monitoring Architecture specified by the OGF

Consumers can use the directory service to discover producers of interest, and producers can use the directory service to discover consumers of interest. Either a producer or a consumer may initiate the interaction with a discovered peer. In either case, communication of control messages and transfer of performance data occur directly between each consumer/producer pair without further involvement of the directory service.

4.2.1 Monitoring and discovering requirements in Computational Grids

Resource discovery and monitoring in large-scale Computational Grids involves determining which computational resources are available to be assigned in order to execute a specific job, application, service, etc., without interfering with their owners' activities. It is very important that Grid Resource Monitoring Systems [Zan05] offer resources availability information reliably, and thus, following resource management phases should be able to keep as balanced as possible resources load through all networks. Therefore, there is an inherent need for novel monitoring systems in charge of sensing computational resources (memory, processor, storage, networking and software applications) as well as monitoring and tracking network-level end-to-end performance.

The challenge and complexity in large-scale Computational Grids is quite different from those in other types of Grids. Due to the fact that the amount of resources required to monitor and discover have been drastically increased, the heterogeneity and diversity found in these systems is greater and more extended. Monitoring systems for large-scale distributed systems have basically three main goals: First, they should report resources availability information when

certain applications or services are running, by means of graphical interfaces or threshold alarms. Secondly, they must support users in finding and keeping track of resources of interest; this is the main purpose of Grid Information Services (GIS) [Cza01]. Finally, they should provide mechanisms to help Grid Resource Management decide which resources certain jobs will be executed in, in order to have a minimum performance impact on the Grid Infrastructure (GI).

A Grid monitoring system that is restricted to a command-line interface may fail to provide easy accessibility to the monitoring information. It is beneficial for the front-end of any grid monitoring system to utilize the client-server model of the Internet, and be able to provide up-to-date information transparently and comprehensively to a client, while performing all the major tasks at the various layers of the backend.

Conclusively, an effective model for a Grid monitoring system should be globally distributed, as decentralized as possible, and capable of providing access to information in a ubiquitous manner. This work describes design, development, and implementation of such a prototype Grid monitoring system.

4.3 Overview of the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling

Large-scale Grid Resource Monitoring and Discovery is addressed by our new SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS). This distributed monitoring system involves a set of autonomous and distributed monitoring agents, which generate real-time and statistical availability information for every resource and entity composing the Grid.

The distributed monitoring agents composing the SBLOMARS approach are pieces of software that act for a user or other program in a relationship of agents and managers (client/server mode). These monitoring agents constantly capture end-to-end network and computational resources performance (processor, memory, software, network, and storage) in large-scale distributed networks. The resources availability information collected is translated to standard reports, which are used by a heuristic resource scheduler to determine a job's requirements and match them to available resources, spreading work between many computers, processes, hard disks, or other resources in order to get optimal resource utilization and decrease computing time around the Grid.

SBLOMARS is already capable of monitoring processor, storage and memory use, network activity at the interface level, services available (an updated list of applications installed), and end-to-end network traffic. This can be done across different architectures, including the Solaris, Unix-based, Microsoft-based, and even Macintosh platforms. Moreover, SBLOMERS is self-extensible and can monitor from multi-processor platforms to huge storage units. Specifically, it is designed around the Simple Network Management Protocol (SNMP) to tackle the generality and heterogeneity problem, and is also based on autonomous distributed agents to facilitate scalability in large-scale Grids.

SBLOMARS addresses the scalability problem by distributing the monitoring system into a set of monitoring agents specific to each kind of computational resource to monitor. Therefore, it deploys a single software thread per type of resource to be monitored, independently of the amount of such resources. This is worthy of mention because many monitoring systems fail when they try to handle new, “hot-plug,” resources that have been added to the system. As we mentioned before, every resource is monitored by independent software threads that start again at certain intervals, becoming an infinite cycle. The cycle-timing is defined by local or remote administrators through booting parameters at the beginning of its execution.

The SBLOMARS monitoring system automatically re-configures its polling periods (calls to SNMP daemon) in an autonomous way. SBLOMARS increases or decreases the time interval between consecutive information requests based on the state of the monitored devices. This

feature is important where the overload caused by any monitoring system is not affordable by the hosting nodes.

The distributed monitoring system reaches a high level of generality by means of the integration of SNMP [Sta99], and thus, offers a wide ability to handle heterogeneous operating platforms. We are aware that integration with SNMP technology is not enough to assure high levels of heterogeneity, but we are presenting a distributed monitoring system design with an exhaustive study of the SNMP incompatibilities. These incompatibilities are related to data formats, objects not implemented, and objects with heterogeneous formats in their data structure. Moreover, this monitoring system integrates an end-to-end network-level monitoring technology that is also consulted by means of SNMP calls. It is the CISCO IOS® IP Service Level Agreements (CISCO IP SLAs) [Cis06], which allows users to monitor end-to-end network-level performance between switches, routers, or from either remote IP network device.

The security of the distributed monitoring system is granted by means of two factors. The first is the fact that SBLOMARS will be deployed on nodes belonging to secure virtual organizations. Then, there is a high level of security from external intruders. Unfortunately, internal users could still modify, intentionality or not, some monitoring parameters on neighbour nodes. Therefore, the second security factor is the implementation of version three of the SNMP protocol (SNMPv2).

SBLOMARS monitoring system has been designed in a flexible manner. This basically means that the monitored information about resources availability is refreshed in self-adjustable intervals of time. It also means that the monitored resources availability information is shown in flexible and dynamic software structures (software structures developed to keep in the memory buffer, for each type of resource, both the amount of used and the amount of available resources since the last polling period), which are used to monitor simple personal computers to complex multiprocessor systems, or clusters with multiple hard disk partitions. Therefore, SBLOMARS is a distributed monitoring system with minimal overload on the performance of the hosting nodes. This feature assures a high level of reliability in computing devices with minimal computational resources, such as mobile devices (i.e. laptops, PDAs, etc).

In summary, our approach is stronger than typical distributed monitoring systems in three essential areas: First, it reaches a high level of generality via the integration of SNMP technology, and thus, we are offering an alternative solution to handle heterogeneous operating platforms. Second, it solves the flexibility problem by implementation of complex dynamic software structures, which are used to monitor simple personal computers to robust multiprocessor systems or clusters, even with multiple hard disks and storage partitions. Finally, it deals with the scalability problem by the distribution of the monitoring system into a set of

monitoring agents, which are specific to each kind of computational resource to monitor. In the Evaluation chapter, we will present the performance tests of the processor and memory overload implied for our resource monitoring and discovering system. These results highlight the fact that the overload of the SBLOMARS monitoring agents is quite affordable. They also show that the SBLOMARS monitoring system offers excellent advantages to resource schedulers, allowing them to perform scheduling in shorter times and distributing the jobs through all available resources in a very fair way.

4.3.1 Design and architecture of the SBLOMARS Monitoring System

In the SBLOMARS monitoring system, each targeted network and computational resource (memory, processor, network, storage, and applications) is considered as an autonomous shared entity. This means that a host/node forming a Grid could be sharing all its computational resources (fully-public), just some kind of them (partially-public), or none of them (fully-private). In Figure 4.2 we show the kinds of resources that the SBLOMARS monitoring agents are able to monitor. It is important to mention that monitoring agents are instances from the SBLOMARS monitoring system, but each kind of resource-specific agent implements specific algorithms to get precise information regarding the resource's usability.

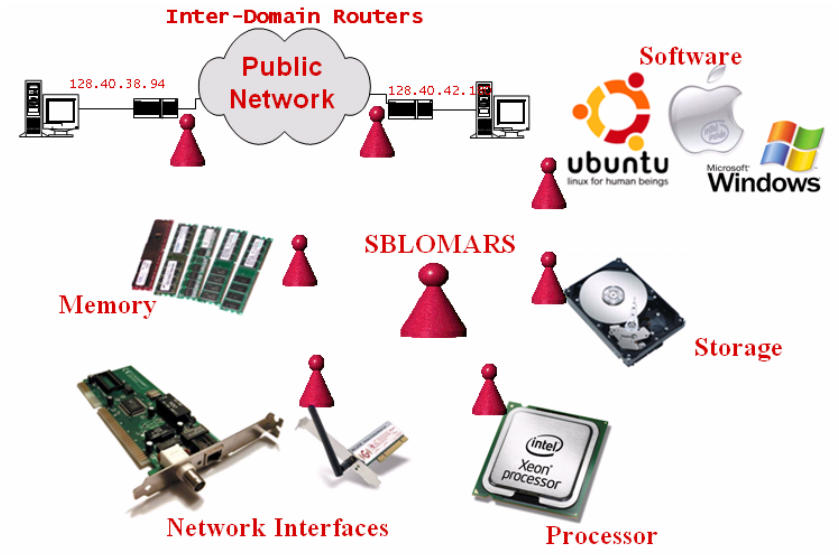

Figure 4.2 SBLOMARS Monitoring Agents distribution per kind of resource

The SBLOMARS monitoring system identifies the shared sources in every node of the Grid and deploys a set of monitoring agents for these specific resources. It could be better explained

with an example. In real organizations, resource owners could share all of their network and computational resources or just some of them. This is possible because SBLOMARS instances a monitoring agent for each resource to share. Therefore, some workstations could offer only memory resources, and other ones could share their storage resources. This is quite common in current distributed organizations. Moreover, a user sharing storage resources could only share part of them. This does not mean that the user is sharing all its storage capacity. For instance, in workstations it is normal to have two hard disk partitions. The operating system is running in the first one, and the user's information is located in the second one. This user will share only the second partition, because he/she does not want to loose the control of the primary partition.

A diagram showing the main components and interfaces of the SBLOMARS monitoring agents is presented in Figure 4.3. We now will describe the functionality of each component and their mutual interactions..

The ***(1)Principal Agent Deployer*** is the main component of the overall monitoring system. It deploys a specific monitoring agent for each kind of resource to be monitored. It offers a generic user interface that can be used to specify the timing between every invocation of the SNMP-MIBs, as well as the number of invocations between every statistical measurement. We have named this functionality as the setting up of the polling periods. Basically, this component starts a monitoring agent for every resource to monitor. For instance, if a certain machine with two hard disks, one micro-processor, two network cards (wireless and wired), and one bank of memory will need six monitoring agents in total, then the Principal Agent Deployer will start six monitoring agents (two for the hard disks, one for the CPU, two for the network cards, and one for memory) with different polling period configurations and, obviously, each monitoring agent will retrieve different OID values from the node SNMP Agents. In chapter 6 (Overall System Evaluation) we will explain this deployment process in a real large-scale scenario.

The ***(2)Resource Monitoring Agents*** component is instantiated in as many classes as there are different kinds of resources that must be tracked. We currently have six different monitoring agents. These are: memory, processor, software, storage, network interfaces (at resource level), and end-to-end network connectivity (at network level). These agents are independent each other due to the fact that SBLOMARS can reduce the overload caused by its monitoring activity when a Grid Node is only sharing some of its resources, but not all of them. In other words, when a workstation is just sharing its storage capacity, SBLOMARS will instantiate only the sub-agent for monitoring storage devices. This ensures that SBLOMARS overload is only what is needed to monitor this resource. Another monitoring alternative based on the deployment of as many monitoring agents as resources discovered on the workstation would cause an unnecessary overload on its hosting nodes.

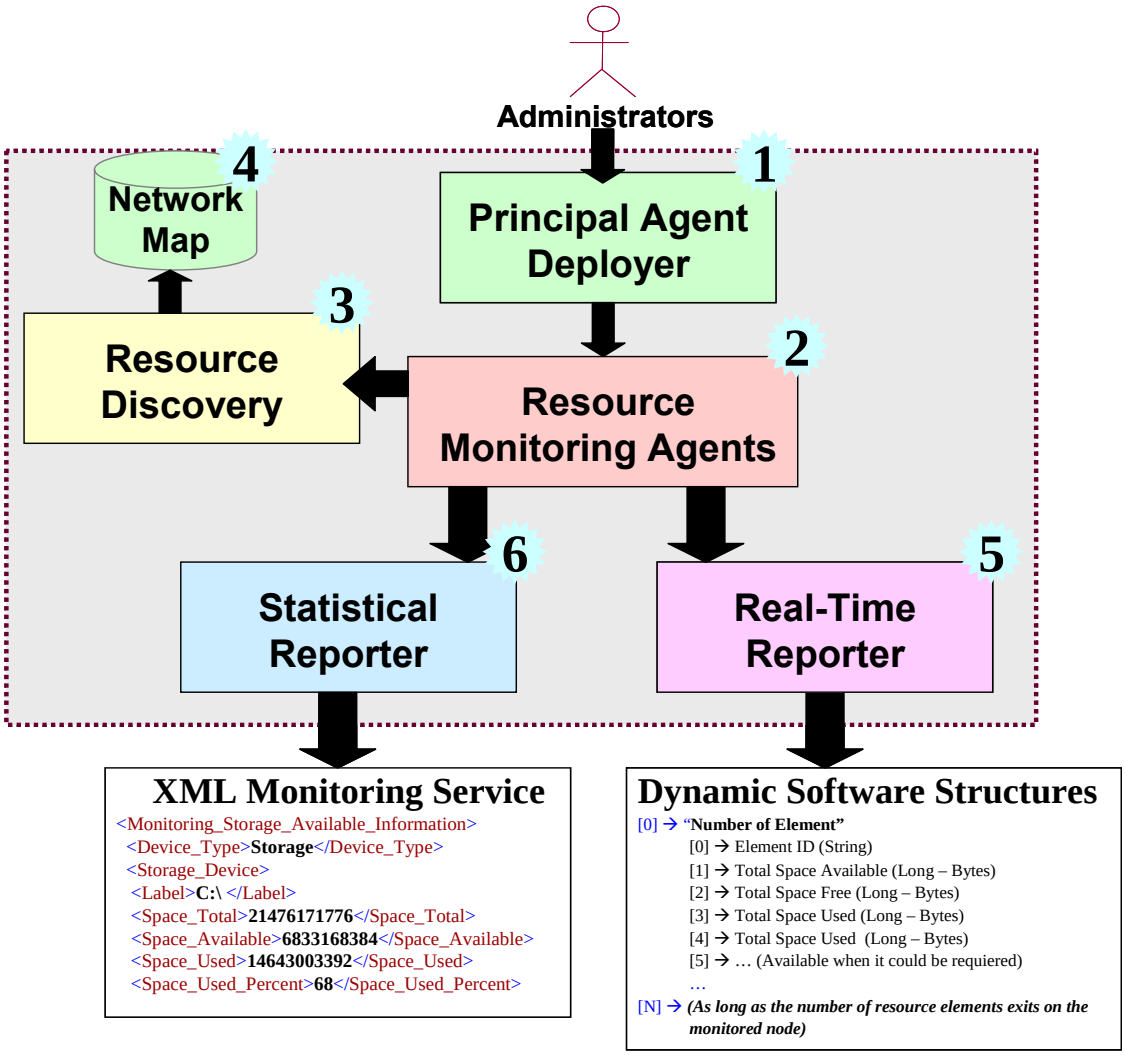

Figure 4.3 The SBLOMARS components and interfaces

The ***(3)Resource Discovery*** component registers the kinds of resources available in the hosting node. We understand as hosting node, the workstation or server where SBLOMARS is being executed. It stores that information in the ***(4)Network-Map Database*** and broadcasts its existence in order for it to be caught by higher level resource schedulers.

The ***(5)Real-Time Reporter*** component generates real-time resource availability information for each kind of resource. It publishes this information by means of ***Dynamic Software Structures*** and offers accessibility to these structures through network socket connections [Yad07]. These structures are available to scheduling systems (e.g. the BLOMERS

scheduler) or any other external instances which need to know the resources behaviour in real-time. We will thoroughly describe dynamic software structures in following sub-sections.

The ***(6)Statistical Reporter*** component generates statistical resource availability information for each kind of resource. It publishes this information by means of the ***XML-based Monitoring Reports,*** and offers accessibility to these structures through network socket connections [Yad07]. The statistical reports are later used in the resource selection phase to determine, in advance and by means of a heuristic approach, which resources are more likely to be the optimal solution for the fulfilment of any user's request.

The SBLOMARS has been implemented in Java [Rus06]. In Figure 4.4 we have depicted a class diagram of the designed core architecture. We have exploited the object-oriented approach that this programming language offers to deploy a single thread per type of resource to be monitored, independent of the amount of such resources. This is worth mentioning because many monitoring systems fail when they try to handle new, "hot-plug," resources that have been added to the system..

Among the most distinguishing aspects of the SBLOMARS in comparison with similar systems we can mention the following. First, it deploys monitoring agents in independent threads for each type of resource to be monitored. This adaptability implies that Grid Nodes may share just some of their resources and not necessarily all of them. Second, the SBLOMARS is able to monitor any amount of shared resources, regardless of software limits (i.e. available memory). This is possible because the SBLOMARS automatically manages its memory buffer to make it as long as necessary or to reduce it when necessary. We can see, then, that deployment of the SBLOMARS monitoring agents is feasible across a wide range of node types, from simple desktop computers to complex multi-processor servers. Lastly, the SBLOMARS self-configures its polling periods (time interval between consecutive retrieval of values from the SNMP-MIBs) automatically. In fact, the SBLOMARS increases or decreases the polling periods according to the state of the monitored resources. We are now going to describe how the SBLOMARS gets realistic and tangible resource availability information from Object Identifiers (OIDs) values from the Simple Network Management Protocol (SNMP) daemon.

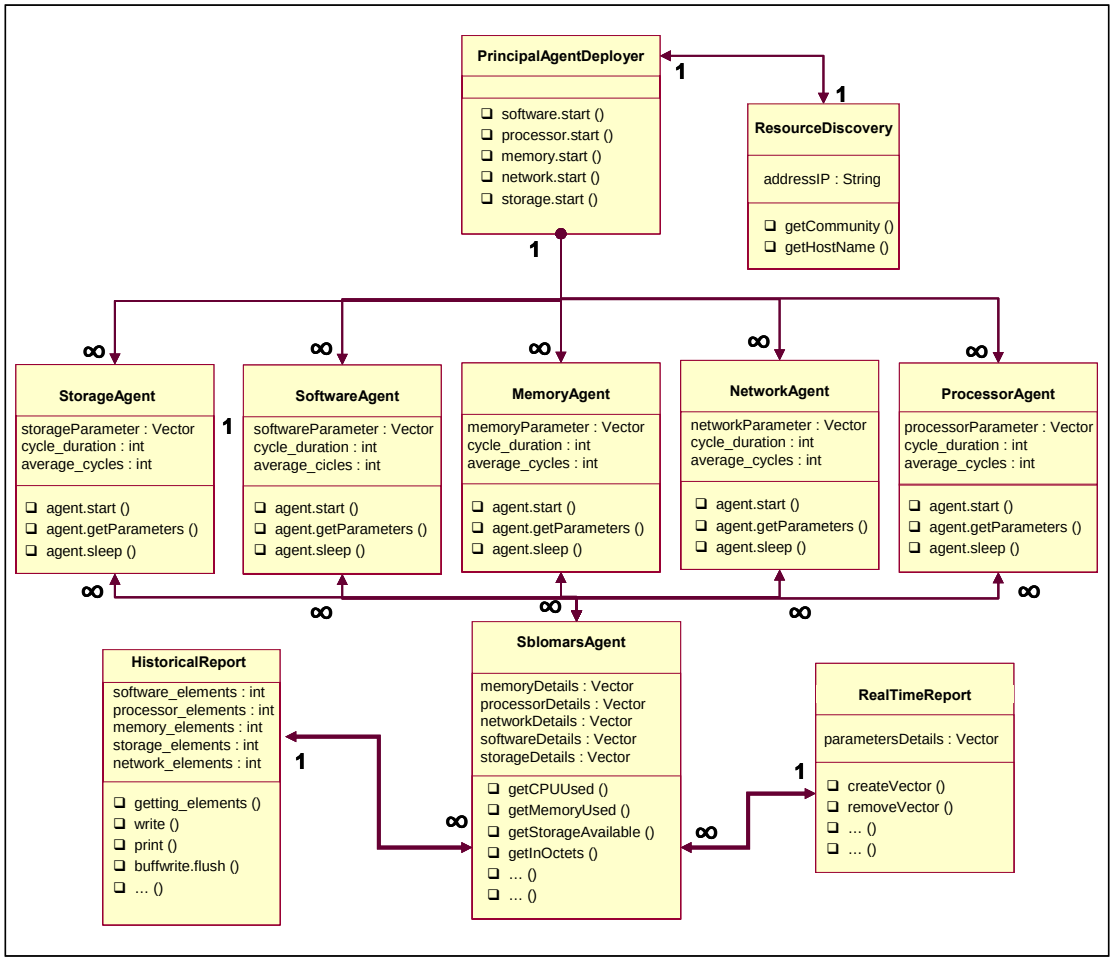

Figure 4.4 Diagram of classes of the SBLOMARS Monitoring System

4.3.2 The SBLOMARS interfaces with SNMP-MIBs

The Simple Network Management Protocol (SNMP) [Sta99] involves nodes acting as management stations (managers) and nodes acting as managed entities (agents). Agents have access to management instrumentation; run a command responder application and a notification originator. A manager contains a command generator and a notification receiver. Management entities control and monitor managed elements. Examples of managed entities are routers or hosts. To convey management information between the managed and the management entities the Simple Network Management Protocol is used, based on the connectionless transport protocol UDP. It is important to note that in the context of our thesis, the role of SNMP managers will be assigned to the monitoring agents instantiated by the SBLOMARS monitoring system.

Monitoring and controlling functions are achieved by requests sent to the SNMP agents and by notifications emitted by the SNMP agents. Management information is abstracted in a Management Information Base (MIB). Objects in MIBs are organized in a way that is described by the Structure of Management Information (SMI). The language used to describe MIB objects is a reduced set of ASN.1 [ANS] constructions. This only allows existence of scalars and two dimensional arrays in the MIBs. Extension of management information is also possible by creating new MIBs or augmenting existing ones. While traditional SNMP-based configuration management enables a device-by-device configuration of network elements, the increasing size and complexity of them has turned configuration into a more difficult task. The increase in size means more devices to configure, the increase in complexity means that devices are of different types, and from different vendors, and perform far more operations.

Our monitoring agents will retrieve the required information by contacting specific objects of the available MIBs. Currently, there are many implemented MIBs around Internet community. Any network/resource element provider implements customized MIBs for their new elements. In the SBLOMARS we have used HOST-RESOURCES-MIB [Sta99], UC-DAVIS-MIB [UCDSN], INFORMANT-MIB [SNINF] and CISCO-RTTMON-MIB [Cis06], which are standard and well-known MIBs for networking and computing resources. The first two mentioned MIBs are configured by default when the SNMP has been installed in the system. The other two are private MIBs, but with open free access. This means that before using the SBLOMARS it is necessary to confirm that these MIBs are already installed in the system. We then ensure that any platform will be monitored by our approach, once it has implemented the above mentioned MIBs. The SBLOMARS also uses alternative open standard MIBs in order to offer full computational resource availability, but it is out of the scope of this thesis to mention all of them.

The most important method in the SBLOMARS is the one to retrieve a specific OID value from the SNMP daemon. Figure 4.5 is a fragment of code as an example of the GET operation from the SBLOMARS monitoring agent to the OID: .3.6.1.2.1.25.6.3.1.2. This OID is not a final value yet, but the root of three where the names of the installed software in the node are allocated. This value could also be accessible by the next identifier: “hrSWInstalledName”.

We have highlighted the GET operation in order to show that it is compatible with the three SNMP versions (1, 2 and 3) and we also want to demonstrate that the SBLOMARS is independent of the OID-MIBs. This means that the SBLOMARS could be improved by adding new functionalities through new MIBs without modifying the core software. It would only be necessary to add the corresponding methods for the new OID values.

```
public String getSoftwareName(int i) {
   String oid = ".1.3.6.1.2.1.25.6.3.1.2."+i;
   String softwareName = "";
   try {
     softwareName =  this.snmpget(version,comm,host,oid);
    …}
   …}

 private String snmpget(
   int version, String comm, String host,   String oid){
     SnmpTarget target = new SnmpTarget();
     if (host != null)
   target.setTargetHost( host );
     if (community != null)
         target.setCommunity( comm );
     if (version != 0) {
   if(version == 2){
      target.setSnmpVersion( SnmpTarget.VERSION2C );
   else if(version == 1)
      target.setSnmpVersion( SnmpTarget.VERSION1 );
   else if(version == 3)
       target.setSnmpVersion( SnmpTarget.VERSION3 );
       else {
        System.out.println("Invalid Version Number");
            }
       }
       target.setObjectID(oid);
       sOut = target.snmpGet();
    return sOut;
    }
```

Figure 4.5 Interface to retrieve MIB Object values by the SBLOMARS Monitoring Agents

The above code is an example to demonstrate the flexibility of the SBLOMARS approach. The SBLOMARS monitoring agents retrieve values from not just one specific OID, but rather from a sequence of OID values. In SBLOMARS just the root of the lists of possible values is set, and then it checks for the full list of values available in the three OID values.

One of the main proposed objectives in the SBLOMARS was to handle a wide variety of operating systems (OS). The simple use of standards like SNMP does not guarantee success in this objective. SNMP implementations are often hampered by a variety of different problems. SNMP managers run into trouble if a device is sending non-standard traps. Although, SNMP is a standard protocol, some people have modified the formats of their traps to suit special needs. They might, for example, have added an extra field to their traps to transmit a particular piece of additional data. If this change was not properly documented, it can cause trouble later. Therefore, it is not possible to ensure that all platforms will be monitored by our approach without explaining the multiple differences and incomplete data between platforms/resources even when standard MIBs are utilized.

The whole set of differences found is too large to describe. Basically, there are three kinds of conflicts or incompatibilities. The first one occurs when MIBs are implemented for certain operating systems, but not implemented for others. Actually, this kind of incompatibility is the most frequent. This is present mainly between Microsoft Windows-based and Unix-based platforms. It is clear that we cannot modify the kernel from all operating systems in order to make our software compatible. The best solution that we have found is to improve our software in order to be adaptable with these issues.

```
public String getMemoryCached() {
String oid=".1.3.6.1.4.1.9600.1.1.2.5.0";
 try {
        memCached =  Integer.parseInt(this.snmpget(oid));
         }
 catch ( Exception ex )
         {
           System.out.println("Exception Getting Memory Cached
           Unix: " + ex.getMessage());
         }
              return memCached;
        }


 public long getMemoryCachedUnix() {
        String oid=".1.3.6.1.4.1.2021.4.15.0";
     try {
        memCached =  Integer.parseInt(this.snmpget(oid));
         }
     catch ( Exception ex )
         {
           System.out.println("Exception Getting Memory Cached
           Unix: " + ex.getMessage());
         }
     return memCached;
 }
```

Figure 4.6 Pseudo code to request heterogeneous MIBs object values

The fragment of code in Figure 4.6 shows how the SBLOMARS solves these kinds of incompatibilities. In this example the SBLOMARS offers the amount of Cache-Memory for Microsoft Windows-based and Unix-based platforms. The getMemoryCached() uses an OID from Informant-MIB [SNINF] and the getMemoryCachedUnix() method uses the UC-Davis-MIB [UCDSN]. This solution is an exhaustive one. We have only highlighted these two platforms because they are the most common ones, but the SBLOMARS offers the amount of Cache-Memory in Mac-based and Solaris-based platforms as well.

The second kind of conflict occurs when the same MIBs show similar requested values in different formats. Even though the two different operating platforms use the same MIB, the content needs to be translated. An example of this would be an application residing in the US

that is communicating with an application in the UK. The date format (from mm/dd/yyyy to dd/mm/yyyy) in the messages between the two MIBs needs to be translated even though they use the same data representation or OID format.

The simplest example of this incompatibility is when the monitoring agents expect to receive ASCII Strings but instead receive Hexadecimal Strings. Thereby, we have implemented the necessary mechanisms to convert Hexadecimal values to ASCII values and vice versa. This solution is not as trivial as it sounds. It was necessary to determine in advance the traps format for ASCII strings in different operating systems, as well as their corresponding versions or distributions. We filtered all received traps until we received an identification string for each available system and then we converted these data strings into standard format by means of a XLM parser. In Figure 4.7 we illustrate the implemented method for the translation of Hexadecimal Strings to ASCII Strings.

The last kind of conflict is found between different distributions, even by the same operating system. For instance, Debian and Ubuntu are similar distributions of Linux. Although in both the HOST-RESOURCES-MIB has been developed, the object getSoftwareName is not implemented. Therefore, there are no alternative options to get this value. On the contrary, Fedora, a similar Linux distribution, has already implemented this object.

In the SBLOMARS we have solved all the above-mentioned issues by means of different actions. Sometimes we have implemented exhaustive mechanisms to cover all possible differences between values or formats and in other solutions we have implemented translators of the OID values formats to generic formats. In the above examples we highlighted each one of the solutions adopted in the SBLOMARS monitoring agents.

```
public String getSoftwareInstalledDate(int i) {
   String oid = ".1.3.6.1.2.1.25.6.3.1.5."+i;
   try {
        softwareInstalledDate = this.snmpget(oid);
        temporal_string = softwareInstalledDate;
        StringTokenizer st = new StringTokenizer(temporal_string);
         while (st.hasMoreTokens())
            {
            String token_string = "";
            if (Label == "") {
              counter = 2;
              while (counter != 0) {
              token_string = token_string + st.nextToken();
              counter --;
            }
             token_string =
```

```
            Integer.toString(Integer.parseInt(token_string, 16));
            }
            else {
             token_string =
             Integer.toString(Integer.parseInt(st.nextToken(),16));
            }
            if (st.hasMoreTokens())
              Label = Label + token_string;
            }
              softwareInstalledDate = Label;
            }
            catch ( Exception ex )
            {
              System.out.println
                  "Problems Getting Software Date: " +
                   ex.getMessage());
                   softwareInstalledDate = "";
            }
        return softwareInstalledDate;
    }
```

Figure 4.7 Pseudo code to request heterogeneous MIB object formats

4.3.3 Configuration of the polling periods in the SBLOMARS

The SBLOMARS monitoring agents are flexible in two aspects: First, the SBLOMARS implements dynamic software structures that have been introduced during the overview section of this chapter. These structures are detailed in sub-section 4.3.4 due to the fact that they are among the standard data interfaces we have designed.

The second aspect where the SBLOMARS reaches flexibility is presented when it re-configures its polling periods automatically. We understand polling period as the rate at which a given MIB object is read to calculate a specific performance metric, such as Storage Capacity (e.g. the total amount of capacity in MB of a hard disk). In fact, the SBLOMARS increases or decreases the time interval between every consecutive polling based on the state of the monitored devices. This feature in the SBLOMARS monitoring system was designed because current monitoring systems for distributed networks such as Ganglia [Mas04] have fixed polling periods, which causes some information regarding resources availability to be lost. This happens when some performance events between consecutive measures are simply not detected.

In our design we have considered this issue. This means that when a resource is being used quite frequently, the time between every trap will be decreased and the amount of monitored information will be much greater. But if a resource is not being used for a long period

of time, the time between every get operation will be increased and the amount of monitored information will be less. Consequently, network overload will be reduced significantly.

4.3.4 Distributed Monitoring Agents data interfaces

As shown in Figure 4.3, the SBLOMARS outputs are presented in two formats: *XML-based Monitoring Reports* containing statistical resource availability information, and *Dynamic Software Structures* containing real-time resource availability information.

The first format, XML-based Monitoring Reports, is designed to be compatible with external systems such as resource schedulers, systems for information forecasting, and resource analyzers. In our general approach, the statistical resource availability reports are used for our heuristic resource scheduler system to determinate in advance which resources have a higher probability of being optimally selected for any users' request.

The statistical reports are developed by means of the XML standard [XML]. Figure 4.8 shows an example corresponding to storage capacity in a workstation with two hard disks, but the second one has two partitions. The SBLOMARS instantiates monitoring agents for each device, so, in this case, there are three instantiated monitoring agents.

In order to generate these reports, the Statistical Reporter component is called by the corresponding monitoring agent to translate the collected data to standard formats. The monitoring agents are in charge of sending the resource availability information from the last polling period to the Statistical Reporter, which collects this information until a previously specified (through Principal Agent Deployer component) number of values are collected, then the Statistical Reporter calculates the average of all the values and generates the XML-based report.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Edited with Agent SBLOMARSXML v1.0 …
<!-- Monitoring Resources Service xmlns:xsi= …
<Monitoring_Storage_Available_Information>
  <Device_Type>Storage</Device_Type>
  <Number_of_Elements>3</Number_of_Elements>
   <Storage_Device>
    <Label>C:\ Label: Serial Number f010b634</Label>
    <Space_Total>21476171776</Space_Total>
    <Space_Available>6833168384</Space_Available>
    <Space_Used>14643003392</Space_Used>
    <Space_Used_Percent>68</Space_Used_Percent>
    </Storage_Device>
  <Storage_Device>
    <Label>G:\ Label:Disco local Serial Number 302e</Label>
    <Space_Total>10733957120</Space_Total>
    <Space_Available>3095842816</Space_Available>
    <Space_Used>7638114304</Space_Used>
    <Space_Used_Percent>71</Space_Used_Percent>
    </Storage_Device>
  <Storage_Device>
    <Label>H:\ Label:SHARED Serial Number 48f893</Label>
    <Space_Total>34290843648</Space_Total>
    <Space_Available>13172244480</Space_Available>
    <Space_Used>21118599168</Space_Used>
    <Space_Used_Percent>61</Space_Used_Percent>
  </Storage_Device>
</Monitoring_Storage_Available_Information>
```

Figure 4.8 XML-based report of storage availability information

The XML-based Monitoring Reports (Figure 4.8) have a great advantage in terms of standardization, but they come at a great computational cost. The activity of creating XML-based files by means of whatever programming language requires considerable CPU usage. Moreover, parsing the content of the information[2] is normally quite slow and also computationally expensive.

The SBLOMARS monitoring agents include an alternative format for interface resource behaviour information. We have designed a set of dynamic structures which are self-extensible regardless the number of new components they involve. We have named this solution *Dynamic Software Structures.* They contain real-time resource availability information. In order to facilitate the scalability of the SBLOMARS monitoring agents, these structures are able to add system components as needed, reducing the load where necessary. The Dynamic Software Structures (Figure 4.9) are software structures that keep in the memory buffer real-time

[2] Parsing is the process of analyzing a sequence of tokens to determine its grammatical structure with respect to a given grammar structure

resource behaviour information from the last polling period. This information remains available until the next resource availability information request. This is another advantage of this approach; any external instance could get this information from the memory buffer in a faster way than accessing the XML-based reports directly. As we have mentioned before, the polling period is assigned by the local or remote node administrator (resource owners) at the moment of configuring the monitoring agents in the Principal Agent Deployer component.

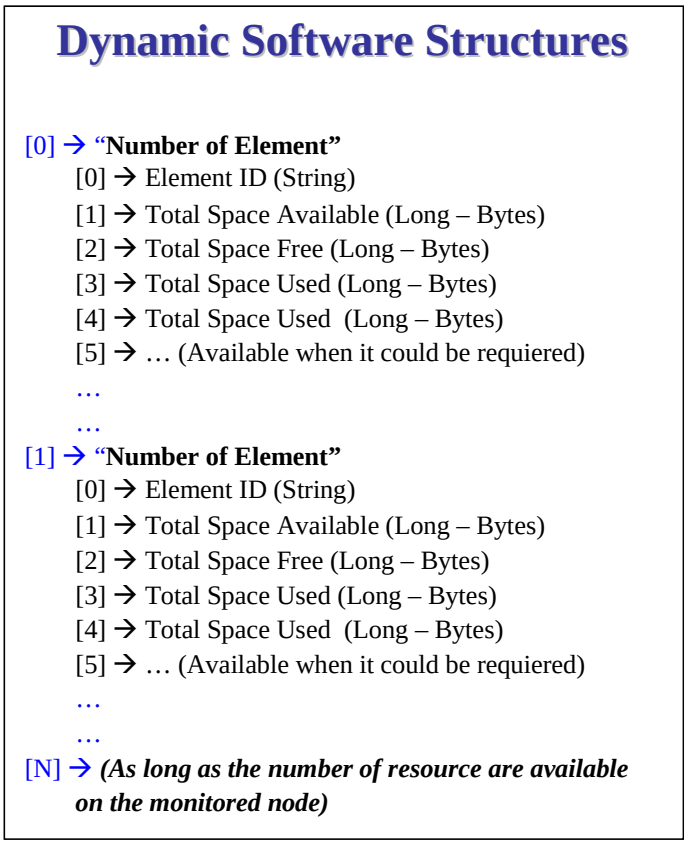

Figure 4.9 Dynamic Software Structure example

The fastest and easiest way to access these structures is through network socket connections. These connections are between the Real-Time Reporter component and any entity consulting this information (e.g. the BLOMERS scheduling system). Another alternative could be by means of CORBA objects or RMI instances. At any rate, defining the implementation is unimportant because any system could adopt the methodology that is best for its own objectives, what is crucial to describe is the ability to add a new server connection for each sub-monitoring agent executing on the host node. This means that SBLOMARS opens a new network socket connection per monitoring agent in order to interface the real-time resource

behaviour information. In Figure 4.9 we show an example of the virtual view from a Dynamic Software Structure about real-time storage behaviour information.

Although, the use of XML standard is well justified in different works [Pra04] and [Kli04], the computational cost is much less when the parsing activity is unnecessary. Our approach is able to handle both XML-based Monitoring Reports and Dynamic Software Structures.

4.3.5 Distributed Monitoring Agents and CISCO IP SLA integration

We have explained how SBLOMARS is able to offer resource availability information about computational resources in a large-scale distributed network, such as the Grid. This information is not enough to decide which resources should be reserved for certain jobs. We also need to know the network performance between nodes running a set of jobs which belong to the same application or service. Moreover, the full process of Grid Services Management will be improved once end-to-end network-level information is included. In order to tackle this important issue, SBLOMARS implements the necessary mechanisms to interface with CISCO IOS® IP Service Level Agreements MIB [Cis06]. This is commonly known as CISCO-RTTMON-MIB (Round-Trip Monitoring MIB).

SBLOMARS offers end-to-end jitter, packet loss rate, bandwidth, and delay average between a Cisco Switch/Router and any IP point. SBLOMARS could also be configured to monitor end-to-end links with different classes of service (DiffServices) [Nic98]. In figure 4.13 we show a snapshot of the web-based SBLOMARS graphical interface showing network information between two Cisco Switches with three different classes of service (VoIP, Multimedia Streaming, and Best Effort).

Basically, Cisco IOS IP SLA uses active monitoring, which includes the generation of traffic in a continuous, reliable, and predictable manner. Cisco IOS IP SLAs actively send data across the network to measure performance between multiple network locations or across multiple network paths, as we show in Figure 4.10. It uses the timestamp information to calculate performance metrics such as jitter, latency, network, and server response times, packet loss, and voice quality scores.

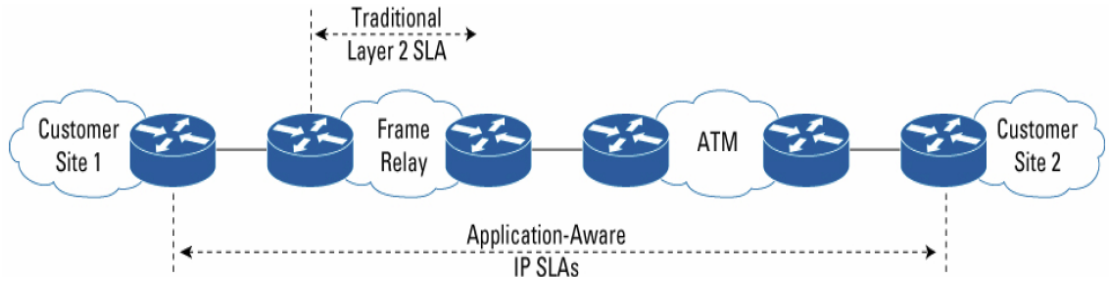

Figure 4.10 Traditional SLAs versus Cisco IOS IP SLAs

The user defines an IP SLA's operation (probe) within Cisco IOS. In Figure 4.12 we show the graphical interface that SBLOMARS includes (this interface has been developed as a part of this research) to create and deploy these probes. In this interface we have included measurement characteristics such as packet size, packet spacing, protocol type, Diff-Serv Code Point (DSCP) marking, and other parameters. The operations are scheduled to generate traffic and retrieve performance measurements. The data from the Cisco IOS IP SLA's operation is stored within the RTTMON MIB and is available for Network Management System applications to retrieve network performance statistics. Cisco IOS IP SLAs is configured to monitor per-class traffic over the same link by setting the DSCP bits. A destination router running Cisco IOS Software is configured as a Cisco IOS IP SLA's Responder, which processes measurement packets and provides detailed timestamp information. The responder can send information about the destination router's processing delay back to the source Cisco router. Unidirectional measurements are also possible using Cisco IOS IP SLAs.

```
public int getJitter(String [] oIDValRTTMon) {
   try {

        jitterSum = sumOfPositiveDS +
           sumOfNegativeDS +
           sumOfPositiveSD +
           sumOfNegativeSD;

        jitterNum = numOfPositiveDS +
           numOfNegativeDS +
           numOfPositiveSD +
           numOfNegativeSD;

        avgJitter = jitterSum/jitterNum;

   …}
…}
```

Figure 4.11 Pseudo code to calculate average jitter by SBLOMARS

Cisco IOS IP SLAs provides a proactive notification feature with an SNMP trap. Each measurement operation can monitor against a pre-set performance threshold. Cisco IOS IP SLAs generates an SNMP trap to alert management applications if this threshold is crossed. Several SNMP traps are available: round-trip time, average jitter, one-way latency, jitter, packet loss, and connectivity tests. Administrators can also configure Cisco IOS IP SLAs to run a new operation automatically when the threshold is crossed.

When a set of probes have been deployed through our SNMP-GUI, CISCO IOS IP SLAs starts to collect metrics regarding network performance and, as we mentioned before, it stores them in the CISCO-RTTMON-MIB. All these values are not functional network monitoring information yet. There are some interpretations and calculations that SBLOMARS has to make in order to get proper networking performance metrics. Our monitoring agents retrieve the fundamental values to get the jitter, delay, packet loss,and bandwidth. We show an example to get Average Jitter. (DS is destination to source, SD is source to destination and the result is given in milliseconds) in the following segment of code in Figure 4.11.

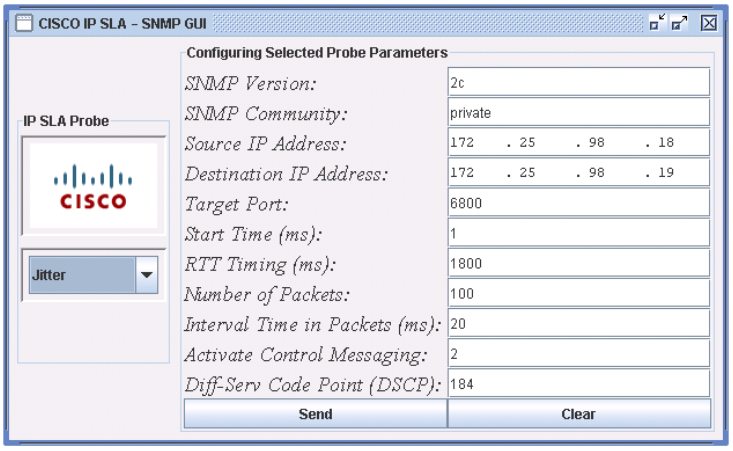

Figure 4.12 SBLOMARS graphical interface with IP SLA probes

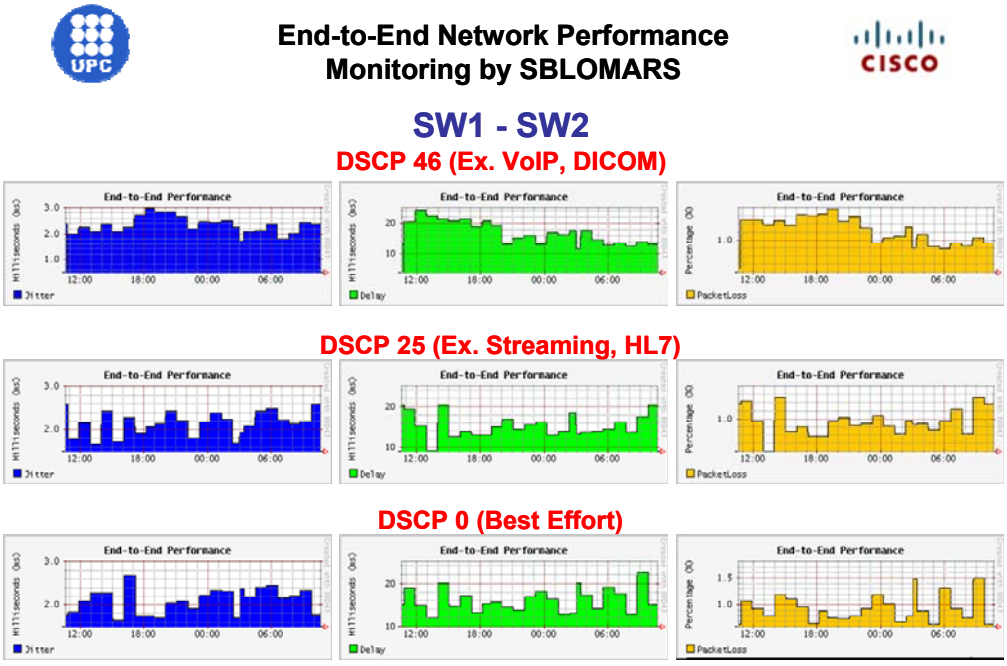

Figure 4.13 Monitoring of three end-to-end service class links by SBLOMARS

4.3.6 Deployment of the SBLOMARS monitoring system

The deployment of the SBLOMARS Monitoring System is done in every node participating into the Virtual Organization (VO). Actually, the Grid is constituted of all these entities, and the VO is just the way they are organized. The distribution of SBLOMARS is done in the following way. The software code is broadcasted along the Grid. Once the nodes have received the code, they start the SBLOMARS monitoring agents, through scripts written into the code itself. This activity is very similar to what occurs when a software disk is inserted into the CD-ROM driver.

When the deployment process has finished, every node forming the Grid is identified by a unique ID. These IDs are stored in a configuration file (network map), which is dynamically updated when new nodes are added or removed from the network. This file is the source of information that the resource scheduler (e.g. the BLOMERS) uses to know which nodes could be asked about their resource availability. The ID format includes an IP Address and a port number. The SBLOMARS monitoring system opens a new port for each resource monitored. Therefore, when a workstation is sharing two hard disks and one processor, the IDs will be as follows (Figure 4.14):

```
Storage_0 → ID_0: 147.83.106.199:6400
Storage_1 → ID_1: 147.83.106.199:6401
Processor_0 → ID_XXX: 147.83.106.199:6600
...
...
```

Figure 4.14 Single node Network Map File example

Regarding identification of the resources located in the same Grid, there is one network map file containing all the IDs from these resources. This file is managed by the resource scheduler. The resulting file has the following structure (Figure 4.15):

```
Storage_0 → ID_0: 147.83.106.199:6400
Storage_1 → ID_1: 147.83.106.199:6401
Processor_0 → ID_2: 147.83.106.199:6600
Memory_0 → ID_3: 147.83.106.145:6200
Storage_0 → ID_4: 147.83.106.145:6400
Processor_0 ( ID_5: 147.83.106.145:6600
Memory_0 ( ID_6: 147.83.106.115:6200
Memory_1 ( ID_7: 147.83.106.115:6201
Storage_0 ( ID_6: 147.83.106.115:6400
Processor_0 ( ID_XX: 147.83.106.115:6600
...
...
```

Figure 4.15 Three nodes (VO) Network Map File example

The process of creating the network map is also triggered by the Resource Scheduling system (BLOMERS). It will be described in detail in the next chapter, when we describe our scheduling approach. We claim that this solution is a hybrid architecture because the monitoring and discovering phase is distributed and the scheduling phase is semi-centralized. Figure 4.16 shows the distribution of our monitoring agents (small red triangular-shaped items inside of any computer) in a real network.

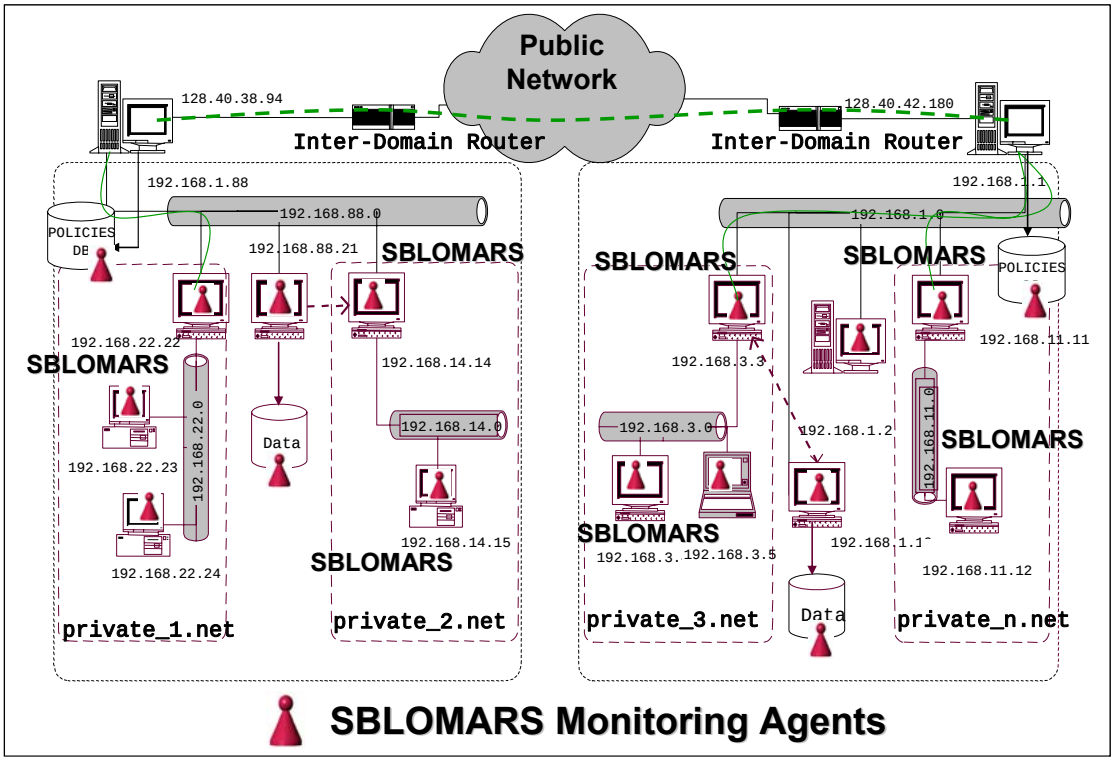

Figure 4.16 The SBLOMARS deployment in two Virtual Organizations

4.4 CONCLUSIONS

In this chapter we have presented SBLOMARS (SNMP-Based Load Balancing Monitoring Agents for Resource Scheduling), an open, free and heterogeneous monitoring system developed for this research work. The SBLOMARS offers a pure decentralized monitoring system in charge of constantly capturing computational resource performance based on distributed agents for multi-constraint resource scheduling in large-scale distributed networks. Also, it has been integrated with CISCO IOS® IP Service Level Agreements (IP SLA) technology, which allows users to monitor end-to-end network-level performance between routers and switches or from either remote IP device, by means of SNMP mechanisms.

The SBLOMARS monitoring system is a set of autonomous and distributed monitoring agents, which generate real-time and statistical availability information for every resource and entity composing the Grid. The SBLOMARS is currently able to monitor processor, storage, and memory use, network traffic at the interface level, services available (an updated list of applications installed), and end-to-end network traffic. This can be done across different architectures, including the Solaris, Unix-based, Microsoft-based, and Macintosh platforms. Moreover, the SBLOMERS is self-extensible to monitor multi-processor platforms to huge storage units. Specifically, it is designed around the Simple Network Management Protocol to tackle the generality and heterogeneity problem, and is also based on autonomous distributed agents to facilitate scalability in large-scale Grids.

The security of the distributed monitoring systems is also tackled. First of all, SBLOMARS is meant to be deployed on nodes which belong to secure virtual organizations. Then, there is a high level of security from external intruders. Unfortunately, internal users could modify, intentionally or not, some monitoring parameters on neighbour nodes. To cope with this problem we rely on the SNMP protocol version 2 (SNMPv2) [Sta99]. Nevertheless, we are aware that security issues are not fully covered to protect this monitoring system from malicious users. This is an area of potential future work.

We have shown some graphical and textual snapshots of resource availability information reports used to represent resources behaviour. However, in Chapter 6 we will present the full set of results obtained from the evaluation trials.

In respect to similar solutions found in the literature, SBLOMARS is a Grid discovery and monitoring system offering high levels of generality through the integration of SNMP technology, and thus, an excellent solution for heterogeneous operating platforms. The SBLOMARS is also flexible because it implements complex dynamic software structures, which are used to monitor from simple personal computers to robust multiprocessor systems, or even clusters with multiple

hard disks and storage partitions. Finally, the scalability problem is covered by the distribution of the monitoring system into a set of sub-monitoring instances which are specific to each kind of computational resource to be monitored (processor, memory, software, network, and storage).

Chapter 5

BALANCED LOAD MULTI-CONSTRAINED RESOURCE SCHEDULER IN LARGE-SCALE GRIDS

5.1 INTRODUCTION

The third phase of the Grid Resource Management (GRM) process proposed in this thesis is Grid Resource Scheduling (GRS). In this phase, when Grid Services need to be allocated on available resources, the scheduling system searches and matches a job's requirements (hard and soft constraints) with resource availability information in order to select the most "appropriate" of them. The term appropriate is related to the proper satisfaction of Grid Infrastructure Providers (GIP) resources usage policies as well as Grid Services Customers (GSC) quality of service policies to maintain high levels of reliability and load-balancing of the Grid Infrastructure.

In other words, it involves assigning computational and networking resources that belong to the GIPs to execute Grid Services across one or multiple administrative domains in a way that minimizes the execution time of a particular Grid Service or a set of them. This period of time that we are working to minimize is commonly known as *makespan*. As we mentioned in the Introduction, Grid Services address fundamental issues in distributed computing relating to how to name, create, discover, monitor, and manage the lifetime of stateful services. Grid Services are defined in terms of standard WSDL (Web Services Definition Language) with minor extensions, and exploit Web Services binding technologies such as SOAP (Simple Object Access Protocol) and WS-Security (Web Services Security).

Grid scheduling is intrinsically more complicated than local resource scheduling because it must manipulate large-scale resources across management boundaries. In such a dynamic distributed computing environment, resource availability varies dramatically. So scheduling becomes quite challenging. Moreover, the Grid Resource Scheduling process should satisfy the Quality of Service level demanded by the GSCs and the resource usage evenness along the whole Grid Infrastructure.

In this chapter we will describe our approach covering the GRS phase in the Grid Resource Management process. In this approach we go one step forward in the state of the art related to current resource scheduler systems, because our proposed system does not just minimize the execution time of a particular set of jobs, but we also propose a resource scheduling system that is capable of balancing resource load through all the entities forming the Grid.

In this chapter we will introduce the Balanced Load Multi-Constrained Resource Scheduler (BLOMERS). This resource scheduling system implements a heuristic approach in order to tackle with the scalability issue. In following sub-sections we will explain in detail the motivation, the architecture, and the operational procedure of the proposed GRS.

The structure of this Chapter is as follows: After this introduction, Section 5.2 briefly describes the Grid Resource Scheduling process, the specific terminology used in this context, and the most important methodologies used to design a scheduling system. Section 5.3 is an overview of our Balanced Load Multi-Constrained Resource Scheduler. In this section we will describe the advantages of using Genetic Algorithms in our scheduling system approach. Then we will describe the design, implementation, functionality, and main features of our heuristic scheduling system. Finally, in Section 5.5 we present the conclusions of this chapter.

5.2 GRID RESOURCE SCHEDULING

In traditional resource scheduling, scheduling is defined as the allocation of jobs to resources over time, taking into account some performance measure criteria subject to the satisfaction of constraints [Don06]. Therefore, we must clarify how Grid scheduling differs from traditional scheduling in order to propose new methodologies or use the current ones.

There is clear evidence that Grid Scheduling differs in fundamental ways from traditional distributed systems resource scheduling. The fundamental difference is the dynamic nature of resources and constraints in the Grid environment; here is where new methodologies with multi-constraints (hard and soft) are needed. Current methods employed to schedule computational resources within the Grid Infrastructure are unsuitable for the demands of a wide variety of potential Grid Services. The enormous facilities and potential of Grids Systems cannot be realised until fundamental development of powerful new scheduling algorithms has taken place. Traditionally, conventional scheduling systems, or batch schedulers are queue-based, and provide only one level of service, namely, "run this when it gets to the head of the queue," which basically means whenever you can do it, just do it.

Developments in Grid Management Systems have presented different scheduling approaches. Here are the most important ones:

I. **Deadline Models:** A user submits a single, self-contained job to the scheduler and wants to get the results of the job back by a certain deadline.

II. **Multi-jobs Composite:** A complex set of sub-jobs must be orchestrated in such a way as to respect any dependence between sub-jobs.

III. **Economical Models:** A user submits a single, self-contained job to the scheduler and wants to get the results of the job back at a certain price.

IV. **Resource Brokers:** A resource broker contacts multiple resources to locate the "best" resource for the needs of a client.

In our approach, resource scheduling is performed based on a Resource Broker methodology. In large-scale distributed computational grids the amount of resources available is extremely large. It is important to know the details of the problem that the scheduling model chosen should solve. Basically, the scheduling problem is described as the selection of “n” number of resources which satisfy hard and soft “k” number of constraints requested by Grid Services’ users. The result of scheduling is a selection of “l-k” resources showing the temporal assignment of operations of orders to the resources to be used.

In our resource scheduling solution, we consider a flexible job shop problem. Each operation can be performed by some machines with different processing times, so that the problem is known to be NP hard. The difficulty is to find the best assignment of a job to a resource in terms of minimizing the total elapsed time (makespan).

The following sub-sections will describe the scheduling problem in detail for large-scale scavenging or distributed computational Grids. Basically, it involves solving for the optimal schedule under various constraints, different resource availability and characteristics of the jobs. We will also describe the analysis process done on this problem during this research in order to justify our heuristic methodology.

5.2.1 The scheduling problem

The scheduling problem is formally represented by a set of jobs J={j1,j2,...,jn} and a set of resources R={R1,R2,...,Rm}. Each job Ji must be performed between a starting time (Ts) and deadline time (Td). The execution of each job requires the use of a set of computational resources (R1, R2, R3, …, RN) which belong to the Grid Environment (Ge), practically a set of nodes N={N1, N2,...,Nn} sharing their resources . The set of nodes and resources available in any Grid Environment (Ge) is represented by the following matrix (5.2.1).

$$\overrightarrow{Ge} = \begin{bmatrix} N_1R_1, N_2R_1, ..., N_nR_1 \\ N_1R_2, N_2R_2, ..., N_nR_2 \\ ..... \\ ..... \\ ..... \\ N_1R_m, N_2R_m, ..., N_nR_m \end{bmatrix} \quad (5.2.1)$$

5.2.2 Searching procedure

The scheduler search procedure is at the core of the scheduling methodology; the procedure examines the set of available resources, generates a number of candidate resources, evaluates the candidate resources to select a final set of resources to be allocated and communicates the results. The procedure takes as input the set of resources (Ge) as well as the scheduling constraints. To find reasonable resources, the search procedure identifies Candidate Resources (Cr) and generates a schedule for resource needs. The final schedule is the best of these candidate resources. The (5.2.2) is the combinatorial number equation; it denotes the

number of ways to choose k items from n items, regardless of order. In the context of this work, n is a number of selected resources that satisfies demanded hard constraints from the whole set of resources (k).

$$Cr = \left(\frac{n}{k}\right) = \left(\frac{n!}{k!(n-k)!}\right) \tag{5.2.2}$$

To guarantee that the optimal Cr will be identified an exhaustive search over all possible unique resource combinations would be required. However, as we demonstrate below, the cost of such a search is prohibitive. In fact, for an exhaustive search, all subsets from size one to the size of the entire resources set must be included in the search. For a resource pool of size "n" the number of distinct Cr that must be included is:

$$Total\_Cr = \sum_{k=1}^{n}\left(\frac{n}{k}\right) = \sum_{k=1}^{n}\frac{n!}{k!(n-k)!} \tag{5.2.3}$$

Two important ways to characterize the effectiveness of an algorithm are its space complexity and time complexity. Space complexity is expressed by the amount of memory that is necessary to execute an algorithm. On the other hand, time complexity deals with determining an expression of the number of steps (computational cost) needed as a function of the problem size (magnitude of the scheduling problem based on the amount of resources available and the amount of constraints to satisfy). Since the step count measure is somewhat coarse, one does not aim at obtaining an exact step count. Instead, one attempts only to get asymptotic bounds on the step count. Asymptotic analysis makes use of the "O" (Big Oh) notation [And00]. The performance of an algorithm is obtained by computing the total number of occurrences of each operation when running the algorithm. This performance is evaluated as a function of the input size "n", and is to be considered as bigger than the multiplicative constants involved in the algorithm equation.

O-Notation gives an upper bound for a function to within a constant factor, as we illustrate in Figure 1. We write f(n) = O(g(n)) if there are positive constants "n" and "c" such that to the right of "n0", the value of f(n) always lies on or below cg(n).

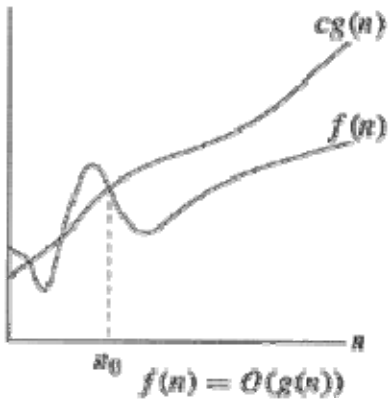

Figure 5.1 O-Notation behaviour graph

The complexity of an algorithm is a function g(n) that gives the upper bound of the number of operations (or running time) performed by an algorithm when the input size is n. There are two interpretations of upper bound.

1. **Worst-case Complexity:** The running time for any given size input will be lower than the upper bound, except possibly for some values of the input where the maximum is reached.
2. **Average-case Complexity**: The running time for any given size input will be the average number of operations over all problem instances for a given size.

Because it is very difficult to estimate the statistical behaviour of the input, most of the time we content ourselves with a worst case behaviour. Most of the time, the complexity of g(n) is approximated by its family O(f(n)) where f(n) is one of the following functions:

- **n** (linear complexity)
- **log n** (logarithmic complexity)
- $\mathbf{n^a}$ where a≥2 (polynomial complexity)
- $\mathbf{a^n}$ (exponential complexity).

In this approach we should find the complexity equation of the scheduling problem representation, in order to find the viability of any solution. In this sub-section we have represented the scheduling problem and we have obtained equation (5.2.3). Now this equation needs to be developed under the O (Big Oh) notation to obtain a representative system for

measuring the performance of our scheduling algorithm. We are assuming that the number of ways to choose "k" items from "n" items, regardless of order, is represented as follows:

$$C\left(\frac{n}{k}\right)\ldots and\ldots\left(\frac{n}{k}\right) = \left(\frac{n!}{k!(n-k)!}\right) \tag{5.2.4}$$

Based on the binomials series [And00], which indicates that:

$$(1+x)^n = \sum_{k=0}^{\alpha}\left(\frac{n}{k}\right)x^n \tag{5.2.5}$$

If we assume that "n" belongs to the positive integer numbers (N) and "x" belongs to the real numbers (R), we obtain the following sub set of equations:

$$(1+x)^n = \left(\frac{n}{0}\right)+\left(\frac{n}{1}\right)x^1+\left(\frac{n}{2}\right)x^2+\left(\frac{n}{3}\right)x^3+\ldots+\left(\frac{n}{k}\right)x^n \tag{5.2.6}$$

If x=1 $$(1+1)^n = \left(\frac{n}{0}\right)+\left(\frac{n}{1}\right)1+\left(\frac{n}{2}\right)1+\left(\frac{n}{3}\right)1+\ldots+\left(\frac{n}{k}\right)1 \tag{5.2.7}$$

$$2^n = \left(\frac{n}{0}\right)+\left(\frac{n}{1}\right)+\left(\frac{n}{2}\right)+\left(\frac{n}{3}\right)+\ldots+\left(\frac{n}{k}\right) \tag{5.2.8}$$

In any scheduling system, it is essential to allocate at least one resource. Therefore is unnecessary to involve the combination when cero resources will be searched. Thus, eliminating the first one of the addition in (5.2.8), we have:

$$2^n - 1 = \left(\frac{n}{1}\right)+\left(\frac{n}{2}\right)+\left(\frac{n}{3}\right)+\ldots+\left(\frac{n}{k}\right) \tag{5.2.9}$$

$$2^n - 1 = \sum_{k=1}^{n}\left(\frac{n}{k}\right) \tag{5.2.10}$$

We note the relationship between (5.2.3) and (5.2.10). Then, we obtain the simplification of (5.2.3) by means of:

$$Total\_Cr = \sum_{k=1}^{n} \left(\frac{n}{k} \right) = 2^n - 1 \qquad (5.2.11)$$

The final step is to apply algorithms analysis in (5.2.11) based on "O" notation as well as its properties. Then, we finally obtain the complexity equation (5.2.13) of the scheduling problem in Grid Environments.

$$g(n) = 2^n - 1 \text{ and } f(x) = O(g(n))$$

$$f(x) = O(2^n - 1) \qquad (5.2.12)$$

$$f(x) = O(2^n) - O(1) \Rightarrow f(x) = O(2^n) \qquad (5.2.13)$$

Once the complexity of an algorithm has been estimated, the question arises whether this algorithm is optimal. An algorithm for a given problem is optimal if its complexity reaches a lower bound over all the algorithms solving the problem. Based on the following table, this is how scheduling in Grid Computing is a NP-hard problem. Moreover, the large-scale distributed computational Grids that we are targeting in this thesis make this algorithm more complex. We have used this analysis in order to look for alternative solutions for this problem.

TABLE 5.1 VALUES OF THE MOST COMMON ALGORITHM COMPLEX FUNCTION

N	Log(n)	n*log(n)	N^2	2^n
1	0	0	1	2
5	0.7	3.5	25	32
10	1.0	10	100	1024
20	1.3	26	400	1,048,576
50	1.7	85	2,500	$1.12590*10^{15}$
100	2.0	200	10,000	$1.26765*10^{30}$
200	2.3	460	40,000	.....
500	2.7	1,349	250,000	.....
1,000	3.0	3,000	1.0*106	.....

For example, to perform a schedule search in a pool set of 20 resources it would be necessary to perform a number of searches given by (5.2.13) for n=20. The resulting value is approximately 1x109. We see that, even for a reasonably sized resource pool, and when the mapping process is time intensive, the enormous size of the search space makes an exhaustive search simply unfeasible. The above table better justifies this statement.

As a conclusion of this algorithm analysis, we can summarize that it is computationally very expensive to analyze the whole set of possible alternatives to find a balanced load solution and also solve an optimization problem for each one. Therefore, we are proposing in this chapter that the Grid Resource Scheduling phase might have to use heuristics to obtain close to optimal solutions in a more efficient manner.

A real environment is more complex than the example we have outlined here. The computation might have a structure that reflects interdependencies among tasks, and we have to consider communication delays for data transfers and coordination overheads. Nevertheless, the example discussed here highlights the importance of implementing an heuristic approach for solving the scheduling problem in large-scale Grids.

The next phase in the presented research methodology is to find a sub-optimal solution to the problem of scheduling resources for grid environments. We have demonstrated that if we have "n" resources, and if a job can be scheduled into any number of resources from 1 to "n", the total number of possible allocations grows exponentially (5.2.11). Thus, it is computationally expensive to analyze all possible allocations and solve an optimization problem for each one. In this case, the Grid's resource scheduler might have to use "heuristics methods" to obtain a near to optimal solution in a more efficient manner.

A wide range of different heuristic methods have been proposed [Rev95], which all have some basic component aspects in common. In general, a representation of partial and complete solutions is required. An objective function is needed, which either estimates the costs of partial solutions or determines the costs of complete solutions. The most crucial component of heuristics methods is the control structure which guides the solution algorithm. Finally, a condition for terminating the iteration process is required. Prominent heuristic methods are, among others, Simulated Annealing [Kir83], Tabu Search [Glo93] and Genetic Algorithms [Bac93]. The first two have been developed and tested extensively in combinatorial optimization. Genetic Algorithms have their origin in continuous optimization. Their theoretical foundation is not well suited for discrete scheduling problems, but in problems with amounts of resources are among the best. Heuristic methods do not guarantee an optimization of the problem proposed, but there are mechanisms to have some estimate of how good or bad a heuristic solution is. This is a matter that has occupied a fair amount of attention in our research, and it is possible to identify at least three techniques to determine the efficiency of heuristic methods, which are Analytical Methods [Fis80], Empirical Testing [Law85] and Statistical Inference [Rev95].

We have used the Analytical and Empirical methods in order to test our scheduling systems. In Chapter 6 (Evaluation) we describe these experiments in detail.

5.3 Overview of the Balanced Load Multi-Constrain Resource Scheduling System

In the proposed research, a new resource scheduling system for large-scale Grids is presented. In the following sub-sections we will describe the motivation, design, and architecture of BLOMERS: Balanced Load Multi-Constrained Resource Scheduler.

This approach works out the third phase of the proposed Grid Resource Management (GRM) process, which is usually named resource scheduling. We have named it Grid Resource Scheduling (GRS). This phase has the responsibility of efficiently matching flexible customer requirements with available resources throughout the Grid in an affordable time, without impacting node performance, and while maintaining the resource load balance.

In large-scale Grids it is very hard to select best suitable resources manually to allocate user jobs in order to achieve a high utilization of the entire set of available resources in the Grid. A scheduler in Grid Computing must cope with system boundaries and manage available resources independent of outside restrictions. The proposed Balanced Load Multi-Constraint Resource Scheduler (BLOMERS) implements a heuristic methodology approach in order to improve the scalability problem and by means of statistical resource availability information, it schedules in a network and resource load-balanced way. Therefore, this approach is not just another resource scheduling system but it is a solution to improve machine utilization with high levels of load balancing. This will be demonstrated in a real large-scale scenario.

The BLOMERS system [Maga07c] deals with several conditions. Basically, it selects a set of candidate resources from a poll, keeping individual resource performance comparatively balanced in all nodes of the Grid. This condition has been added in order to satisfy computational resource load balancing. In the BLOMERS approach, we propose a sub-optimal solution to the problem of scheduling computational resources in large-scale Grids, whose distinguishing feature is that it makes use of a Genetic Algorithm (GA) .

5.3.1 Genetic Algorithms in Large-scale Computational Grids

Genetic Algorithms (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. They are a particular class of evolutionary algorithms (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes, genotype or the genome) of candidate solutions

(called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. A more detailed description and explanation of the theory behind GAs is described on Appendix B.

In Genetic Algorithms (GAs) each chromosome represents a solution, using strings of 0's and 1's. Each bit typically corresponds to a gene. This is called binary encoding. The values for a given gene are the alleles. A chromosome in isolation is meaningless. We need to decode the chromosome into phenotypic values. In Figure 5.2 we show the before-mentioned components used on the GA area. Figure 5.3 shows an example of the evolution of the populations from a generation N to generation N+1.

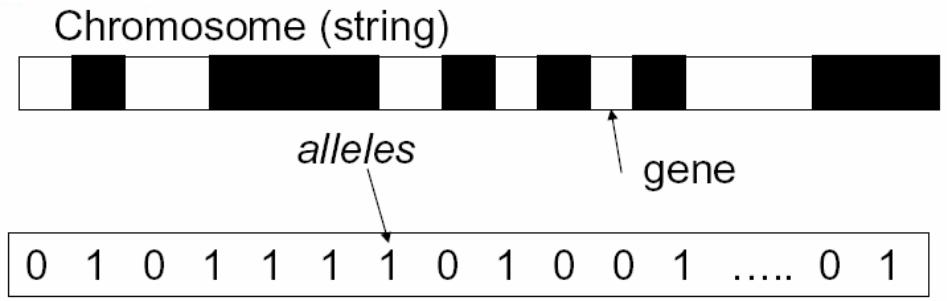

Figure 5.2 Codification of the resource ID by means of binary code

GA differs from traditional search/optimization methods [Gar00] in the following aspects:

- GAs search a population of points in parallel, not only a single point.
- GAs use probabilistic transition rules, not deterministic ones.
- GAs work on an encoding of the parameter set rather than the parameter set itself.
- GAs do not require derivative information or other auxiliary knowledge - only the objective function and corresponding fitness levels influence search.

There are well known differences between Genetic Algorithms and other heuristics approaches [Rev95]. We have chosen GAs as our heuristic solution in the BLOMERS system because this kind of algorithm is a search algorithm based on the conjecture of natural selection and genetics. The features of genetic algorithm are different from other search techniques in several aspects. First, the algorithm is a multi-path that searches many peaks in parallel, hence reducing the possibility of local minimum trapping. Secondly, GAs work with a coding of parameters instead of the parameters themselves. The coding of a parameter will help the

genetic operator to evolve the current state into the next state with minimum computations. Thirdly, GAs evaluates the fitness of each string to guide its search instead of the optimization function. GAs only needs to evaluate objective function (fitness) to guide its search. There is no requirement for derivatives or other auxiliary knowledge. Finally, GAs explore the search space where the probability of finding improved performance is high.

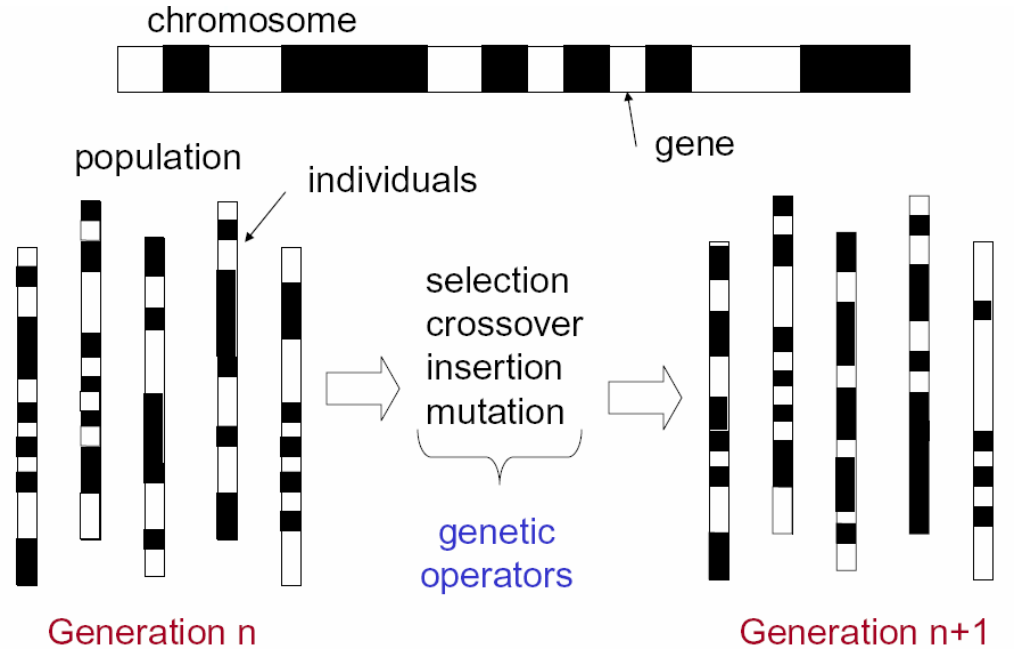

Figure 5.3 Evolution of Populations in Genetic Algorithms

The key step in the any Genetic Algorithm-based solution is the evolution mechanism. It is the moment when new populations (set of candidate solutions) are created by the algorithm. The BLOMERS uses probabilistic rules to evolve a population from one generation to the next. The generations of the new solutions are developed by two genetic recombination operators:

- **Crossover:** Combining parent chromosomes to produce children chromosomes. Crossover combines the "fittest" chromosomes and passes superior genes to the next generation. Crossover takes parent-pairs from selection step and creates a new population.
- **Mutation:** Altering some genes in a chromosome. Mutation ensures the entire state-space will be searched, (given enough time) and can lead the population out of a local minima. Mutation makes "slight" random modifications to some or all of the offspring in the next generation.

The use of both genetic recombination operators helps our resource scheduler approach to fairly distribute jobs with available resources. Crossover is applied when resources from the same domain are requested and Mutation is used when resources from different domains are also requested. Using both operators we have found it easier to balance the resources load in the Grid Infrastructure. Moreover, the originality of any GA implementation is found in these operators. The internal algorithms for performing the new population mechanisms (Crossover and Mutation) are completely different from one design to the other. These are the tuning components of the Genetic Algorithm to be customized for any specific approach. Therefore, during this research we have worked on the implementation of these algorithms, and we will thoroughly describe them in following sub-sections. We will also describe detailed examples of these operators in our resource scheduler system.

Finally, it is worth mentioning that BLOMERS scheduling system needs both real-time and statistical resource availability/behavioural information. Otherwise, it will not be functional at all. The BLOMERS obtains this information from SBLOMARS (monitoring agents), but that does not mean that BLOMERS simply accepts data from SBLOMARS. It could receive the same kind of resource availability/behavioural information from alternative monitoring systems like the Globus MDS [Bor05] or Ganglia [Mas04].

5.3.2 The BLOMERS Genetic Algorithm Design

In this section we will describe our approach to perform the Grid Resource Scheduling (GRS) phase in Large-scale Grids. We now come to the third phase in the proposed Grid Resource Management process.

In the BLOMERS approach, we propose to find a sub-optimal solution to both scheduling computational resources and load-balancing resources problems in large-scale distributed computational Grids. The algorithms for reduction of the makespan and fair load along all involved resources are truly new contributions at the Grid Resource Management area; we consider these approaches as novel functionalities in the scheduling process. This scheduling process is part of the Grid Resource Management System, which is embedded into a Policy-based Grid Resource Management Architecture, which has been thoroughly described in Chapter 3.

The genetic algorithm for resource selection has to deal with several conditions [Fos99]. Basically, it should select a set of candidate resources from a poll, keeping individual resource performance comparatively equal in all nodes of the distributed system. This condition has been added in order to satisfy computational resource load balancing. Finally, the resource selection

algorithm needs to keep the relative operations' sequences, known as precedence constraint of the type i → j. This constraint is defined as: data generated by task i are required to start task j.

Despite what most people believe, Genetic Algorithms are a very simple methodology to solve NP-hard problems. The flowchart for our algorithm design is shown in Figure 5.4. Basically, there are two fundamental activities in this flowchart. The first one is the valuation of the generated population and the second one is reproduction into a new population.

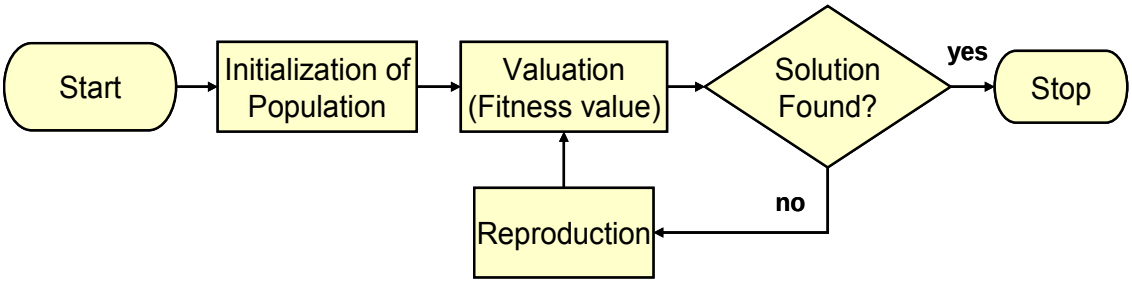

Figure 5.4 The Flowchart of BLOMERS Genetic Algorithm Design

The pseudo code of the heuristic resource scheduling algorithm and its description are as follows:

```
CleaningBuffer (Pk)
Initialize (k, Pk);
Evaluate (Pk);
     Do
       {
       Select_Resource_Candidates (Pk);
       Crossover (Pk);
       IF Evaluate (Pk+1) == Minimal Constraints;
          Ends Do-While;
       ELSE
          Mutation (Pk);
       IF Evaluate (Pk) == Minimal Constraints;
          Ends Do-While;
          }
Deliver (k_solution);
```

Figure 5.5 BLOMERS genetic algorithm pseudo code

BLOMERS uses a collection of solutions (population) from which better solutions are produced using selective breeding and recombination strategies. The first population is created randomly by means of the ***Initialize (k, Pk)*** method. Where k is the kind of resource to analyze

and Pk is the population of k resources selected. The presented genetic algorithm works with several threads in parallel, one for each kind of resource available. SBLOMARS monitoring agents offer a socket connection per resource monitored. We will describe the BLOMERS algorithm in general because each software thread has the same architecture and functionality.

The method ***Initialize (k, Pk)*** calls a random method to select the first population of our solution. BLOMERS is implemented in JAVA code; we have used the primitive random function in this method. Once the first population has been initialized, a first simple evaluation of this population is done. Normally, the first population is never selected as a candidate solution, but it is the main entry to create the new populations. The ***Select Resource Candidates (Pk)*** method bounds the initial populations and applies two simple genetic operators, such as ***Crossover (Pk)*** and ***Mutation (Pk)***. These methods are used to construct new solutions from pieces of the old one in such a way that the population (Pk) steadily improves.

5.3.3 Encoding Solution of the BLOMERS Genetic Algorithm.

In the design of our Genetic Algorithm into the resource scheduler we first need to show the code used to represent our resources in the Grid Infrastructure. In Section 5.3.1 we explained that GAs works with coding of parameters instead of real values. In BLOMERS every kind of resources performs a selection independently of the other kinds of resources. This means, for instance, that storage resource availability information is analyzed in a different parallel software thread than the software thread analyzing memory resource availability information. This programming philosophy is very similar to the one used in the design and implementation of the SBLOMARS monitoring system.

Moreover, in Chapter 4 we describe how the SBLOMARS monitoring agents represent resources through IDs. These IDs are stored in a configuration file (network map), which is dynamically updated when new nodes are added or removed from the network by SBLOMARS monitoring system itself. BLOMERS scheduling system uses this network map file and translates it into as many lists as kinds of resources available (memory, storage, processor, etc.).

The best way to explain the encoding design of the Grid resources is through the following example. In Figure 5.6 and Figure 5.7 we show XML-based reports generated by SBLOMARS monitoring system. In these reports we clearly see all available shared resources in two Pentium IV workstations. The first one has 512MB of RAM memory, one hard disk with two partitions, and Windows XP as the Operating System. The second one has 1GB of RAM memory, one hard disk with only one partition, and Linux Ubuntu 6.10 as Operating System. We

show these reports in order to explain how SBLOMARS and BLOMERS interface with each other and translate this information into resource identifiers (IDs).

```
<?xml version="1.0" encoding="UTF-8" ?>
 <!-- edited with Agent SBLOMARS XML v1.0 (http://nmg.upc.edu/~emagana) by TSC (UPC)  -->
 <!-- Monitoring Resources Service xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  -->
<Report_Period_Time> 10
   <Operative_System>  WindowsXP
      <Monitoring_Information>
         <Device_Type> Storage
                  <Number_of_Elements>2</Number_of_Elements>
                  <Storage_Device>
                   <Label>C:\ Label: Serial Number f010b634</Label>
                   <Space_Total>21476171776</Space_Total>
                   <Space_Available>9658118144</Space_Available>
                   <Space_Used>11818053632</Space_Used>
                   <Space_Used_Percent>57</Space_Used_Percent>
                   </Storage_Device>
                  <Storage_Device>
                   <Label>G:\ Label:Disco local Serial Number 302e56e2</Label>
                   <Space_Total>10733957120</Space_Total>
                   <Space_Available>1011720192</Space_Available>
                   <Space_Used>9722236928</Space_Used>
                   <Space_Used_Percent>91</Space_Used_Percent>
                   </Storage_Device>
                   </Device_Type>
                  <Device_Type> Processor
                   <Number_of_Elements>1</Number_of_Elements>
                  <Processor_Device>
                   <Kind>Hardware: x86 Family 6 Model 6 Stepping 2 AT/AT COMPATIBLE</Kind>
                   <Percentage_Used>9</Percentage_Used>
                   </Processor_Device>
                   <Device_Type>Memory</Device_Type>
                   <Number_of_Elements>1</Number_of_Elements>
                  <Memory_Device>
                   <Kind>RAM</Kind>
                   <Memory_Total>1073205248</Memory_Total>
                   <Memory_Available>578220032</Memory_Available>
                   <Memory_Used>358166528</Memory_Used>
                   <Memory_Used_Percent>40</Memory_Used_Percent>
                   </Memory_Device>
                   </Device_Type>
                  <Device_Type> Network_Interfaces
                   <Number_of_Elements>1</Number_of_Elements>
                  <Network_Device>
                   <Label>Adaptador Ethernet 3Com 3C900COMBO</Label>
                   <Speed>10000000</Speed>
                  <In_Octets>3591930</In_Octets>
                  <Out_Octets>1117971</Out_Octets>
                  <BW_Percentage_Used>0</BW_Percentage_Used>
                  </Network_Device>
                  </Device_Type>
         </Monitoring_Information>
      </Operative_System>
   </Report_Period_Time>
```

Figure 5.6 Workstation A - XML-based Report Generated by SBLOMARS

```
<?xml version="1.0" encoding="UTF-8" ?>
 <!-- edited with Agent SBLOMARS XML v1.0 (http://nmg.upc.edu/~emagana) by TSC (UPC)  -->
 <!-- Monitoring Resources Service xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  -->
<Report_Period_Time> 10
   <Operative_System>  Linux_Ubuntu 6.10
      <Monitoring_Information>
         <Device_Type> Storage
                  <Number_of_Elements>1</Number_of_Elements>
                  <Storage_Device>
                   <Label>/ root</Label>
                   <Space_Total>21476171776</Space_Total>
                   <Space_Available>9658118144</Space_Available>
                   <Space_Used>11818053632</Space_Used>
                   <Space_Used_Percent>43</Space_Used_Percent>
                   </Storage_Device>
                   </Device_Type>
                  <Device_Type> Processor
                   <Number_of_Elements>1</Number_of_Elements>
                  <Processor_Device>
                   <Kind>Hardware: x86 Family 6 Model 6 Stepping 2 AT/AT COMPATIBLE</Kind>
                   <Percentage_Used>9</Percentage_Used>
                   </Processor_Device>
                   <Device_Type>Memory</Device_Type>
                   <Number_of_Elements>1</Number_of_Elements>
                  <Memory_Device>
                   <Kind>RAM</Kind>
                   <Memory_Total>1073205248</Memory_Total>
                   <Memory_Available>578220032</Memory_Available>
                   <Memory_Used>358166528</Memory_Used>
                   <Memory_Used_Percent>46</Memory_Used_Percent>
                   </Memory_Device>
                   </Device_Type>
                  <Device_Type> Network_Interfaces
                   <Number_of_Elements>1</Number_of_Elements>
                  <Network_Device>
                   <Label>Adaptador Ethernet 3Com 3C900COMBO</Label>
                   <Speed>10000000</Speed>
                  <In_Octets>21930</In_Octets>
                  <Out_Octets>21930</Out_Octets>
                  <BW_Percentage_Used>0</BW_Percentage_Used>
                  </Network_Device>
                  </Device_Type>
      </Monitoring_Information>
   </Operative_System>
</Report_Period_Time>
```

Figure 5.7 Workstation A - XML-based Report Generated by SBLOMARS

Figure 5.8 is a representation of the network map file considering that only these two workstations are our Grid Infrastructure. Actually, in this table we show four files, BLOMERS scheduling system generates one network map file per kind of resource that is discovered in collaboration with SBLOMARS monitoring system.

STORAGE
ID_0: 147.83.106.199:6400
ID_1: 147.83.106.199:6401
ID_2: 147.83.106.201:6400
…

PROCESSOR
ID_0: 147.83.106.199:6000
ID_1: 147.83.106.201:6000
…

MEMORY
ID_0: 147.83.106.199:6200
ID_1: 147.83.106.201:6200
…

NETWORK INTERFACES
ID_0: 147.83.106.199:6800
ID_1: 147.83.106.201:6800
…

Figure 5.8 List of Shared Resource from Workstation A and Workstation B

The encoding of resource availability information is based on Figure 5.8. In our approach we substitute these referenced IDs (0, 1, 2, …, N) by a 10-bits binary code. The binary code used is just the binary representation of the ASCII number which represents the IP address and the port of the resource available within the Grid Infrastructure. In order to maintain the scalability of the scheduler, we have implemented a dynamic coding system. Therefore, each Chromosome represents a resource, using strings of 0's and 1's. The Population is the number of Chromosomes available to test. In Figure 5.9 we illustrate a coding example from the STORAGE network map file from Figure 5.8.

STORAGE	CHROMOSOME
ID_0: 147.83.106.199:6400	0000 0000 00
ID_1: 147.83.106.199:6401	0000 0000 01
ID_2: 147.83.106.201:6400	0000 0000 02
…	…

Figure 5.9 Encoding Example of Storage Resource from Workstations A and B

5.3.4 Crossover Operation

Here, selected individuals are randomly paired up for crossover (sexual reproduction). This is further controlled by the specified crossover rate and may result in a new offspring individual that contains genes common to both parents. New individuals are injected into the current population. Crossover operators in our approach exchange substrings of two parents to obtain two offspring. In BLOMERS scheduling system the crossover operator combines useful parental information to form new and hopefully better performing offspring. Such an operator can be implemented by choosing a point at random, called the crossover point, and exchanging the segments to the right of this point. For example, let:

			CHROMOSOME
ID: 24	Parent 1	=	0000 0110 00
ID: 49	Parent 2	=	0000 1100 01
…	…		…

Figure 5.10 Example of the Random Selection of Two Chromosomes Population

And suppose that the crossover point has been chosen randomly as indicated by the red colon. The resulting offspring would be:

			CHROMOSOME
Storage_0 → ID_24: 147.83.106.199:6400	Parent 1	=	0000 0110 00
Storage_1 → ID_45: 147.83.106.167:6401	Parent 2	=	0000 1100 01

Figure 5.11 Example of Population Evolution Before Crossover Operation

And suppose that the crossover point has been chosen randomly as indicated by the dark-red colon. The resulting offspring would be:

			CHROMOSOME
Storage_1 → ID_25: 147.83.106.199:6401	Child 1	=	0000 0110 **01**
Storage_0 → ID_44: 147.83.106.167:6400	Child 2	=	0000 1100 **00**

Figure 5.12 Example of Population Evolution After Crossover Operation

In the example shown in Figures 5.11 and 5.12, Child 1 and Child 2 will be the resulting population. The advantage of using the proposed crossover operation is that we can select individuals from the same Virtual Organization. Here, the selection mechanism takes into account neighbour nodes when they are not processing high amounts of tasks. We illustrate this operator by the algorithm in Figure 5.13.

```
Crossover (Pk);
     {
      Start While
      Choose randomly two parents (chromosomes).
      Extract the last bit from Parent 1
      Extract the last bit from Parent 2
      Exchange last bit Parent 1 with last bit Parent 2
      Substitute the last bit of Parent 1 with last bit of Parent 2
      Store the resulting chromosome as Child 1
      Substitute the last bit of Parent 2 with last bit of Parent 1
      Store the resulting chromosome as Child 2
      Check with SBLOMARS the last average value of Child 1
      If last average value is less than 30% (QoS Policy)
           Then, Child 1 is chosen.
      Otherwise, change Parent 1 and Start While
      Check with SBLOMARS the last average value of Child 2
      If last average value is less than 30% (QoS Policy)
           Then, Child 2 is chosen.
      Otherwise, change Parent 1 and Start While
      End while.
      }
Deliver (k_solution);
```

Figure 5.13 Crossover Operation Algorithm Implemented in BLOMERS

5.3.5 Mutation Operation

In this step, each individual is given the chance to mutate based on the mutation probability specified. If an individual is to mutate, each of its genes is given the chance to randomly switch its value to some other state. In BLOMERS resource scheduler algorithm the mutation operator randomly alters each Chromosome with a small probability, typically less than 1%. This operator introduces innovation into the population and helps prevent premature convergence on a local maximum.

The evolution is terminated when the population attains certain criteria, such as simulation time, number of generations, or when a certain percentage of the population shares the same function value. In our approach a clear example is as follows:

			CHROMOSOME
Storage_1 → ID_25: 147.83.106.199:6401	Parent 1	=	0000 0110 01
			MUTATION
Storage_N → ID_57: 147.83.**206**.199:6401	Child 1	=	0000 **1**110 01

Figure 5.14 Example of Population Evolution with Mutation Operation

In this example, Child 1 will be the resulting population. The difference from the previous operation is that the mutation operation helps to move from one Virtual Organization or Administrative Domain in the Grid towards others with similar resource availability information. In Grid 5000 test-bed, we have applied the mutation operation to move from one cluster (Lyon) to the next one (Bordeaux). In this part, we present a new mutation operator, called the controlled mutation operator, designed especially for our parallel jobs encoding, as it can balance the machine loads. The algorithm of this operator is presented In Figure 5.15.

```
Mutation (Pk);
     {
     Start While
     Choose randomly one chromosome (Parent 1)
     Select the bit 5 and 6 from the chosen chromosome
     If bit == 0
          Then, bit = 1;
     Otherwise
          bit = 0;
     If last average value is less than 30% (QoS Policy)
          Then, Child 1 is chosen.
     Otherwise, change Parent 1 and Start While
     End while.
     }
Deliver (k_solution);
```

Figure 5.15 Mutation Operation Algorithm Implemented in BLOMERS

5.3.6 Selection Mechanism

Here the performance of all the individuals is evaluated based on the fitness function, and each is given a specific fitness value. Here it is important to mention that the higher the value, the bigger the chance of an individual passing its genes to future generations. Through the overview section of this Chapter, we have mentioned that Genetic Algorithm could apply several

selection methods. In BLOMERS scheduling system we have chosen the Elitism selection, in order to distribute jobs as fairly as possible.

In the proposed and implemented Elitism selection method, we have chosen a group of individuals from a population. This group is composed of seven individuals that are selected at random from the population, and the best (normally only one, but possibly more) is chosen. The best is the one that uses, at the moment of the scheduling, the least amount of its shared resources.

This methodology is considered an exhaustive search. In this case, the exhaustive methodology for scheduling does not consume exponential times, as we have demonstrated in previous sub-sections, because the number of resources to compare is restricted to the offspring population. This population, as we said before, is composed of only seven Chromosomes (candidate resources).

Fitness is a measure of how well an algorithm has learned to predict the outputs from the inputs. The goal of the fitness evaluation is to give feedback to the learning algorithm regarding which individuals should have a higher probability of being allowed to multiply and reproduce, and which of them should have a higher probability of being removed from the population.

This algorithm is faster than other heuristic methodologies, and could be better adapted to heterogeneous parameters, but the most important advantages are that it avoids failing into a local minima solution, and it can be run in parallel. We made use of this advantage to design our genetic algorithm in many threads as many resource devices have been found on the VO.

Our approach is not as effective as exhaustive search approaches, but it is faster and lighter in terms of computing performance impact. Basically, every genetic algorithm has a selection probability (Ps) function based on its “fitness value” for each solution. The coefficient between the fitness value of a specific solution and the summary of all the fitness values of the same one is the selection probability for this solution:

$$Ps(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)} \qquad (5.3.1)$$

Applying this methodology is not a simple task. The genetic algorithm needs to be adapted according to the requirements of the application and the environment in which it will be working. The information to analyze for each search will be adaptable to resource availability and the conditions for this adaptation are completely different in each design, which ensures the novelty and originality of this approach.

5.3.7 Interface between BLOMERS and SBLOMARS

It is clear that one of the most important activities to perform the full Grid Resource Management (GRM) process is to create the network map of the discovered available shared resources. The network map creation process is triggered by the BLOMERS scheduling system. Each monitoring agent is in charge of reporting its activity to BLOMERS. In this process BLOMERS has an open socket connection which listens to all agents running on the Grid Infrastructure. The objective of this mechanism is to create a database of all agents that are live on the Grid. This database is the first reference to create the network map file once the resource scheduler starts to ask the monitoring agents about their resource availability. The pseudo code shown in Figure 5.16 illustrates the implemented interface between SBLOMARS and BLOMERS systems.

```
  public RegisteringProcessorAgents () {
     try {
        System.out.println("Listening in Port: " + ... );
        listen_socket = new ServerSocket(resource_manager_port);
        registering = new ProcessorAgentsFile ();
        }
     catch (IOException e) {
     System.out.println("Exception Creating Server Socket: ");
       }
  }

  public void thread_Calling_Sockets () {
     try {
       while (true) {
       System.out.println("Registering of Processor Agents");
       Socket skCliente = listen_socket.accept();
       InputStream in = skCliente.getInputStream();
       ObjectInputStream receive = new ObjectInputStream(in);
       Object received = receive.readObject();
       String iPAddress = received.toString();
       System.out.println("Processor Agent: " + iPAddress);
       registering.generate_vector(iPAddress);
       skCliente.close();
       in.close();
       receive.close();
     }
  } catch( Exception e ) {
     System.out.println( e.getMessage() );
     }
  }

SortingVector (String iPAddress, Vector _directions) {
       this.ipAddress = iPAddress;
       this.directions = _directions;
       }

public void thread_Sorting_Vectors() {
```

```
      System.out.println(ipAddress);
      int flag = 1;
      if (!directions.isEmpty()){
      for (int i=0; i<directions.size(); i++) {
            String checking = (String)directions.get(i);
            if (checking.compareTo(ipAddress) == 0) {
            flag = 0;
            break;
      }
      }
            if (flag == 1)
            directions.addElement(ipAddress);
      }
      else
            directions.addElement(ipAddress);
            callingVector (directions);
    }

public void callingVector (Vector _directions) {
      ProcessorAgents returnVector = ProcessorAgents ();
      returnVector.refreshingVector (_directions);
     }
```

Figure 5.16 BLOMERS Scheduling System Code for Sorting Resources IDs Values from Multiple SBLOMARS Monitoring Agents.

It is worth mentioning that the **SortingVector** method is crucial in this interface. This is because the amount of resource references, actually the IP Address of every resource with the corresponding port to access its availability information, is quite large, and the activity of sorting this database is very important. We have implemented the bubble sorting methodology for this interface. This methodology is very efficient in sorting such structural information. The justification for the implementation of this methodology is the need to avoid repeating the references of resources that have already registered in the database. It This is a common problem because in order to have a discovery system that is reliable we must have register calls sent from SBLOMARS monitoring agents to BLOMERS resource scheduler quite often. Then we ensure that our approach will be aware of resources that have recently been integrated with the Grid Infrastructure, and also the BLOMERS scheduler will be aware of the resources that are not more available to be taken into account in the process of generating the new population of the Genetic Algorithm.

In Figure 5.17 we illustrate a typical scenario where BLOMERS scheduling system is interfacing with SBLOMARS monitoring agents. The solution presented in this thesis is a hybrid architecture, the monitoring and discovering phase is a distributed solution where monitoring agents (SBLOMARS) are spread along the network, and the scheduling phase is a semi-

centralized solution where Genetic Algorithms (BLOMERS) are performing matches of resources with available load-balancing resources.

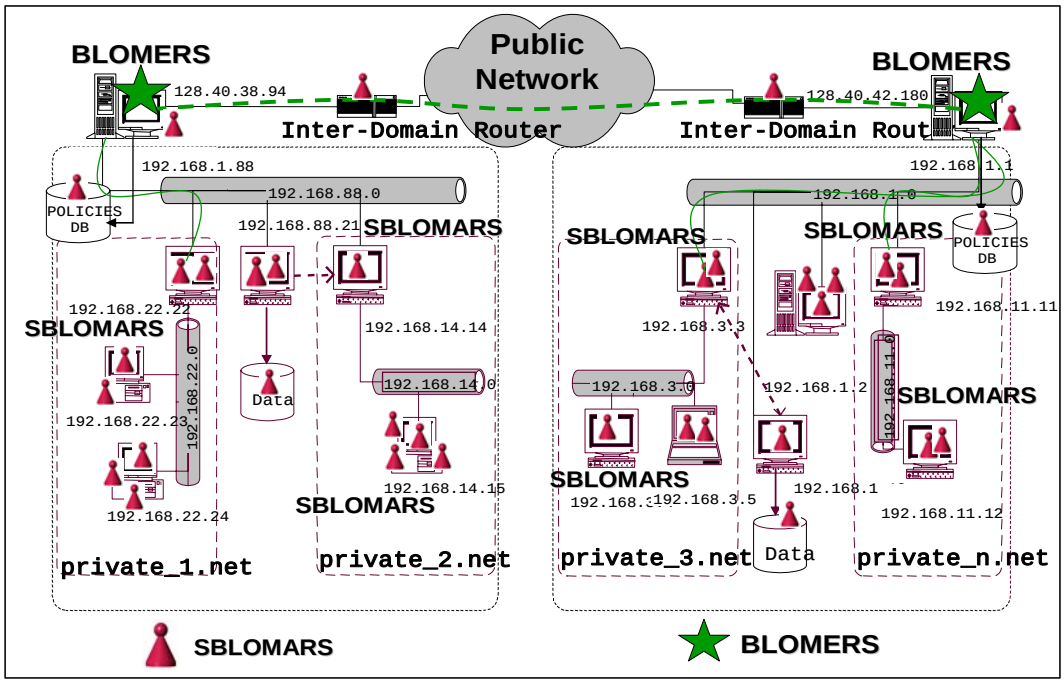

Figure 5.17 BLOMERS Distribution in Virtual Organizations

The network map has a virtual view for the SBLOMARS monitoring agents as follows:

Storage_0	**ID_0**	**147.83.106.199:6400**
Storage_1	**ID_1**	**147.83.106.199:6401**
Processor_0	**ID_2**	**147.83.106.199:6600**
Memory_0	**ID_3**	**147.83.106.145:6200**
Storage_0	**ID_4**	**147.83.106.145:6400**
Processor_0	**ID_5**	**147.83.106.145:6600**
Memory_0	**ID_6**	**147.83.106.115:6200**
Memory_1	**ID_7**	**147.83.106.115:6201**
Storage_0	**ID_8**	**147.83.106.115:6400**
Processor_0	**ID_9**	**147.83.106.115:6600**
…	**ID_N-1**	**…**
…	**ID_N**	**…**

Figure 5.18 Example of a Network-Map File View by BLOMERS

This information is collected by BLOMERS resource scheduler through socket interfaces with SBLOMARS monitoring agents. Therefore, it is very important to understand how the mechanism works to collect all this information and to create the corresponding network map file. There is only one network map per virtual organization or sub-network, as we have illustrated in Figure 5.18.

We show the pseudo code for the socket interface implemented by SBLOMARS monitoring agents. This socket interface is redundantly implemented by each agent running on the hosting node. Thereby, we would have as many socket interfaces as resources available on the hosting node. The pseudo code for this interface is illustrated in Figure 5.19.

```
public String Sblomars_Interface_BLOMERS (String node) {
     try{
       String [] finalValues = new String[3];
       HOST = node;
       System.out.println( "Inicia Socket Client" );
       Socket skCliente = new Socket( HOST , PUERTO );
       System.out.println("Calling SBLOMARS by BLOMERS");
       System.out.println("Calling to: " + HOST);
       System.out.println("Using Port : " + PUERTO);
       InputStream in = skCliente.getInputStream();
       ObjectInputStream receive = new ObjectInputStream(in);
       Object received = receive.readObject();
       Vector newvalue = new Vector () ;
       newvalue = (Vector) received;
       finalValues = (String []) newvalue.get(0);
       percentage = finalValues [2];
       System.out.println(percentage);
       skCliente.close();
       in.close();
       receive.close();
      }
      catch( Exception e ) {
      System.out.println( e.getMessage() );
  }
  return percentage;
 }
```

Figure 5.19 Interface Code Implemented in SBLOMARS Monitoring System

On the other hand, there should be an entity or interface in charge of contacting with the socket interfaces implemented by the monitoring agents in order to collect the resource availability information. In this interface, there should be only one instance per resource to request availability information. Therefore, the same interface could work for all the resources available on the Grid Infrastructure. The pseudo code for this interface is shown in Figure 5.20.

```
public int getting_port ()
   {
      int port = 0;
      GettingPorts ports = new GettingPorts ();
      try {
         //Configuration Files
         String config = "ports.conf";
         String resource = "processor";
         port = (ports.initFromFile(config, resource));
         }
     return port;
   }

public void refreshingParameters (Vector parameters) {
      //System.out.println("New Parameters");
      cpuParameters = parameters;
   }

public void closingSocket () {
      //System.out.println("Closing Socket Memory");
      try {
         listen_socket.close();
        }
      catch (Throwable e) {
      System.out.println("Exception Socket:" e.getMessage());
      }
   }

public void requestingValues Thread() {
    try {
       while (true) {
       //System.out.println("Parametros CPU");
       Socket skCliente = listen_socket.accept();
       OutputStream out = skCliente.getOutputStream();
       ObjectOutputStream sendStream = ObjectOutputStream(out);
       Vector v = proc_agent.cpuParameters;
       sendStream.writeObject(v);
       skCliente.close();
       out.close();
       sendStream.close();
       }
   }
}
```

Figure 5.20 Interface Code Implemented in BLOMERS Scheduling System

The most important method from the above extracts of the BLOMERS scheduling system interface with SBLOMARS monitoring system is ***requestingValues***. In this method BLOMERS is able to contact the monitoring agent in charge of reporting availability information for a specific resource (i.e. amount of memory free in the cluster university.edu).

This interface is the one most frequent called by the Genetic Algorithm. This is because the GA needs real-time resource availability information in order to calculate the fitness value for this specific resource. Therefore, this interface is called several times by the algorithm.

5.4 CONCLUSIONS

In summary, scheduling computational resources in Grid Computing is a matter that requires significant innovations to improve the efficiency of the actual mechanisms implemented, mainly because the Grid behaviour is constantly variable, resources availability is always unpredictable, their performance is highly unstable, and the amount of resources to compute is undetermined in most cases. The current methods employed to schedule jobs on computing resources within the Grid are unsuitable for the demands of a wide variety of potential Grid applications. The enormous potential of the Grid cannot be realized until fundamental development of powerful new scheduling algorithms has taken place. In this thesis we have investigated and developed a novel scheduling methodology that address the critical issues of flexibility and re-negotiation of user requests, and seeks to address the problem of handling uncertainty and imprecision in both computing resources and user requirements. The heuristic methodology presented will be exploited in order to use and apply the most appropriate of its advantages to achieve better performance of the Grid Resource Management process (GRM) in the environment of a Grid Management Architecture based on policies.

In this chapter we have described the main features and design details of our resource scheduler approach to solve the inherent NP-hard problem of scheduling computational resources in large-scale Grids.

We have illustrated the need for implementing alternative solutions of the job-shop scheduling problem for large-scale Grids. In a real scenario where plenty of resources are available and Grid Services have complex requirements, the solution was complex, and we have demonstrated that it is a typical NP-hard problem with the following equation to calculate the computational cost of the exhaustive solution:

$$f(x) = O\left(2^n - 1\right)$$

It is extremely expensive, in terms of computational cost, to find the optimal solution for the scheduling problem. Therefore, we have looked for alternative solutions which, though considered sub-optimal, carry a much lower computational cost than the previous well-known heuristic methodologies.

The solution described in this Chapter is known as the Balanced Load Multi-Constrained Resource Scheduler (BLOMERS). BLOMERS implements a Genetic Algorithm, which offers a parallelism to multi-constraint service requests avoiding enclosure into local minima.

BLOMERS solves the Grid Resource Scheduling (GRS) phase of the Grid Resource Management (GRM) process proposed in this thesis. In this phase, the scheduling system searches and matches hard and soft constraints with resource availability information in order to select the most “appropriate” of them.

It covers multi-constraint (hard and soft constraints) service requirements, which are: user requirements (QoS, deadlines, etc.), service needs (memory, storage and software requirements), and resource-load balancing throughout the Grid.

In our solution the most appropriate resources to be selected are those that satisfy the Grid Services hard and soft constraints, which we have detailed in this research. Also, the secondary policy for selection of resources is the balancing of the resources’ usability in the whole Grid.

We have described how the BLOMERS system implements two mechanisms to generate new populations. The first is by means of the mutation mechanism. This mechanism helps our scheduler to generate an offspring which is in the same sub-network as its parents. The motivation of this mechanism is to search resources that are shared on workstations relatively close to the one previously chosen. Because the probability of similar resources on workstations in the same sub-network is quite high(I don’t understand this sentence and so can’t edit it...) that looking for these resources into different sub-networks.

The second one is the crossover mechanism. This mechanism helps our scheduler to generate an offspring which is not in the same sub-network as its parents. It could be the case that the resources used in a determinate sub-network are highly selected and the whole Grid could be unbalanced. Therefore, we have implemented the crossover mechanism to guarantee that shared resources from external sub-networks, which can be also different virtual organizations, will also be considered in the fitness process.

Therefore, BLOMERS improves resource load-balancing and reduces the makespan in any scheduling. It is an important contribution of our research because the most important approaches on this area only focus on reducing the makespan of the scheduling process but do not maintain resource usability in a balanced way.

BLOMERS exhibits excellent performance. It is able to schedule large numbers of services in real scenarios, and guarantees a balanced load throughout the Grid. The makespan measured is less than those of the round-robin and least used algorithms. In Chapter 6 we will show a graphical comparison of these three algorithms.

Chapter 6

OVERALL SYSTEM EVALUATION

6.1 INTRODUCTION

In previous chapters, we have described the four proposed phases in the GRM process. We have also explained that these phases require the intervention of three main systems, which are the Policy-based Grid Resource Management Architecture (PbGRMA), the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS), and the Balanced Load Multi-Constrained Resource Scheduler (BLOMERS). In this Chapter we will provide a set of evaluation experiments for these three systems. Once the evaluation of each system is performed, we will evaluate the full performance of the general approach presented in this thesis into the Grid5000 [Grid5000], which is a real, large-scale test-bed.

Regarding the SBLOMARS monitoring system, we will show the low overhead caused by its agents on the hosting workstations or servers. There is also a demonstration of the flexibility of this approach as well as its heterogeneity. We have run some experiments to demonstrate how scalable SBLOMARS monitoring system is, and finally we will show the amount of storage resources consumed by the XML-based reports (statistical resources availability information) in twenty-four hour, seven day experiments.

We will show the advantages of the BLOMERS scheduling system compared with alternative scheduling algorithms and we will estimate the average fitness of individuals (groups of available resources) matching a required schema (constraints for resource load balancing and minimizing makespan). A similar performance evaluation of BLOMERS is also included as the performance evaluation done for SBLOMARS. An analytical evaluation will be presented in order to demonstrate how good the BLOMERS scheduling system is compares with classical selection mechanisms such as random selection.

The Policy-based Grid Resource Management Architecture will be evaluated in this chapter in two parts. The first is intended to show how well the PbGRMA performs in isolation. This means that we will run some performance tests to check the reliability of the system when it is executing the Grid Services Management (GSM) phase of the GRM process. We obtain positive results by measuring the elapsed times during the process of merging the three sources of constraints (quality of service needs, computational-networking resource availability, and Grid Services specifications) which form the Domain-Level Grid Service Policy.

The second part of the PbGRMA evaluation is intended to show the full GRM process functionality in a real large-scale scenario. In this case we will be able to demonstrate how the Jobs Allocation and Activation (JAA) phase is executed. The scenario is the Grid 5000 test-bed. Section 6.2 is an overview of the Grid 5000 architecture and functionality details. The evaluation of the full GRM process will help to support our claims regarding the reliability, heterogeneity, and scalability of the four-phase PbGRMA presented in this thesis.

The structure of this chapter is as follows: Subsequent to this Introduction, in Section 6.2 we briefly describe the Grid 5000 test-bed and the workflow required to deploy experiments on it. Section 6.3 presents an evaluation of the SBLOMARS monitoring system. In Section 6.4 we present the evaluation of the BLOMERS scheduling system. In Section 6.5 the evaluation of the PbGRMA is presented in isolation and also running in the Grid 5000 test-bed. In Section 6.6 we present the conclusions of this chapter.

6.2 THE GRID 5000 TEST-BED DESCRIPTION

One of the requirements of this research was to execute the Grid Resource Management process in a real, large-scale scenario. Therefore, we decided to do our testing in an experimental test-bed, namely the Grid5000 [Grid5000]. This is a French Ministry of Research Grid initiative partially financed by INRIA, the ACI GRID incentive action, CNRS, and RENATER.

Large-scale distributed systems like Grids are difficult to study only from theoretical models and simulators. Most Grids deployed at large scale are production platforms that are inappropriate research tools because of their limited reconfiguration, control, and monitoring capabilities. Nevertheless, Grid5000 is designed to provide a scientific tool for computer scientists similar to the large-scale instruments used by physicists, astronomers and biologists.

This platform currently involves 3000 nodes located in 10 different sites in France. These nodes are linked through 1 and 10Gbits links (Figure 6.1). We evaluated our approach on different Grid scenarios (micro Grid of nodes geographically located on one site, Enterprise small-scale Grids with few dozens of nodes located on a reduced number of sites, and large scale Grids with 10 sites and a few hundred nodes). Different time frames were also considered (from a few hours to a few days).

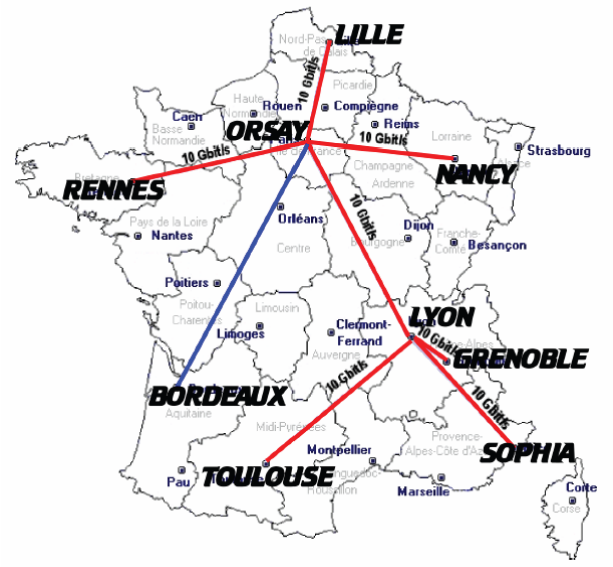

Figure 6.1 Number of Nodes and Sites in the Grid5000 Test-bed 1

[1]This figure was taken from www.grid5000.fr

The Grid5000 workflow is depicted in the below graph (Figure 6.2). This sequence of activities is the same for every user of the test-bed. In our case we followed this workflow to set our experiments up. The transmission of our experimental parameters is done by "SSH" connections. The execution and the collection of the information are also done by means of SSH connections. There is no other alternative due to security mechanisms implemented by the administrators of the Grid5000.

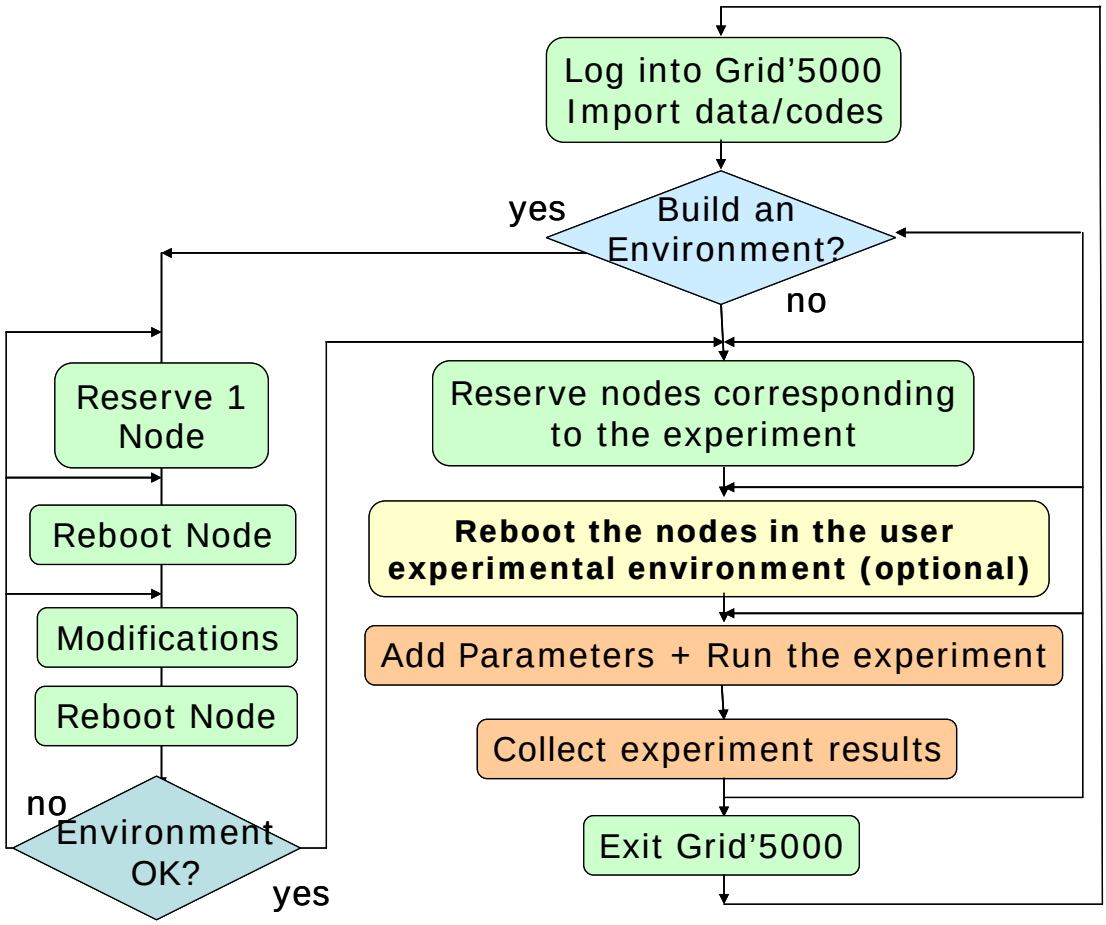

Figure 6.2 Grid5000 workflow for deploying experiments

In the first instance, we have configured our Grid Infrastructure by means of two basic tools available in the Grid 5000 test-bed. The first is the OAR 2.0 [OAR] software, which is a resource manager (or batch scheduler) for large clusters. It allows cluster users to submit or reserve nodes either in an interactive or a batch mode. This tool was used to reserve our nodes from the rest of nodes available in Grid5000. It is wothy of mention that this test-bed is in use by many Universities and research centers, therefore the number of experiments running every day is

quite high. The second tool used in this experiment was the KADEPLOY 2.1.5 [KAD]. It is a fast and scalable deployment system for cluster and grid computing. It provides a set of tools for cloning, configuring (post installation), and managing a set of nodes. It currently successfully deploys linux, BSD, Windows, Solaris on x86 and 64 bits computers.

We need both tools because nodes available in Grid5000 need to be reserved, configured, and finally deployed. In order to understand the set of steps followed to set up the Grid scenario for our experiments we will describe these steps in detail. It is important to highlight that GRID5000 is a collection of nodes from nine sites in France. Due to the fact that Gridd5000 is a private test bed, there is only one way to access these nine sites, and that is through a cluster front-end per site. This design guarantees the security of the full test bed. From this front-end any authorized user can reserve any amount of resources available, but must follow some administrative policies. Therefore, the set of steps are as follows:

I. **Cluster Front End connection:** In this step, client access at any of the nine available clusters through a Secure Shell (SSH) connection. The names of these clusters are as follows:
    a. Lyon
    b. Bordeaux
    c. Grenoble
    d. Lille
    e. Nancy
    f. Orsay
    g. Rennes
    h. Sophia
    i. Toulouse

II. **Perform Nodes Reservation:** Once the authorized user is remotely connected to one of the available clusters. It is necessary to deploy a reservation command by means of the OAR tool. A reservation is a time slot in the future that we want to reserve to then connect to once it's time. The following command reserves 12 nodes for 5 hours starting at 11am on April 27, 2007: It is important to highlight that resources are transparent for the final user in Grid Coputing philosophy. Grid 5000 testbed was designed under the same fundamentals, therefore there are not necessity for the final users to know which exactly nodes are using for executing their Grid Services

```
# oarsub -r "2007-04-27 11:00:00" -l walltime=5, nodes=12
```

III. **Checking Environments:** When a reservation is performed, the client receives a notification regarding the status of the reservation. The client also receives a reservation ID, which will be used in further processing. In order to know the names of the nodes reserved, it is possible to have a list of these nodes. This list is allocated in the following variable:

```
$OAR_FILE_NODES
```

IV. **Deploying Nodes:** This is the final step to complete both processes, the reservation and configuration on the Grid5000 test-bed. In this step the KADEPLOY tool is used to deploy a pre-selected configuration of the Operating System that is required to start the individual experiments. It means that every node will be running the configuration of the operating system selected for these experiments.

```
# kadeploy -e environment_name -m node -p partition_device
```

So far, the set of nodes reserved is ready to perform any experiment that the client wants to deploy.

6.3 SBLOMARS EVALUATION

SBLOMARS is an approach by which software agents are constantly capturing end-to-end network information (jitter, packet loss ratio, delay, etc.) and computational resources performance (processor, memory, software, network and storage) in large-scale distributed networks[2].

The distributed monitoring agents approach monitors resource behaviour, but only in local resources. This means that SBLOMARS agents must be installed in workstations or servers which have been required to be monitored. Therefore, performance evaluation of the monitoring agents is very important. In the following sub-section we will describe the experiments to evaluate the overload of SBLOMARS in terms of memory, cpu cycles, and number of live threads deployed. These are typical performance metrics used to analyze the performance of software components.

6.3.1 Performance Evaluation

The objective of this test was to analyze the processor and memory consumption of the SBLOMARS software package on its hosting node. Also, we were interested to observe the self-management capability of its software threats. Consequently, we proceeded installing the SBLOMARS in a Pentium IV with 512MB of RAM and Ubuntu Linux 6.10 OS.. The following results are reported by means of the Java Profiler [JAPRO]. This programmer's tool can obtain a variety of system's information, among which we can find our data of interest; Java Profiler helps software developers find performance bottlenecks, pin down memory leaks, and resolve threading issues.

Figure 6.3 is a twenty-four hours snapshot for the percentage of CPU usage by SBLOMARS in its hosting node. The SBLOMARS package and Java Profiler run at the same time on the same workstation. It does not matter that other applications could be running on the hosting node because Java Profiler is configured to only measure resource usage by SBLOMARS.

As can be observed, SBLOMARS represents an insignificant overload in the hosting workstation. During a twenty-four hour experiment, the SBLOMARS average CPU usage was

[2] In Chapter 4 of this thesis, we have described the motivation, design, and architectural details of the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS).

7,78% and crossed an arbitrary set threshold of twenty percent of total CPU capacity of the workstation only five times.

The CPU usage peaks shown in Figure 6.3 were due to self-reconfigurations of the SBLOMARS agents polling times. In he first, second and fifth peaks, the polling time was increased, whereas in the other two it was decreased. When the polling period is increased, the amount of information generated per resources is decreased. When the polling period is decreased the amount of resource availability information generated is proportionally increased. Therefore, peaks on CPU performance are presented when the monitoring system performs these re-configurations. SBLOMARS CPU percentage usage normally oscillates between one and eight percent. In System performance could be briefly affected, but in general there is not noticeable overload. A proper evaluation of the reconfiguration functionality in SBLOMARS monitoring system is presented in section 6.3.2.

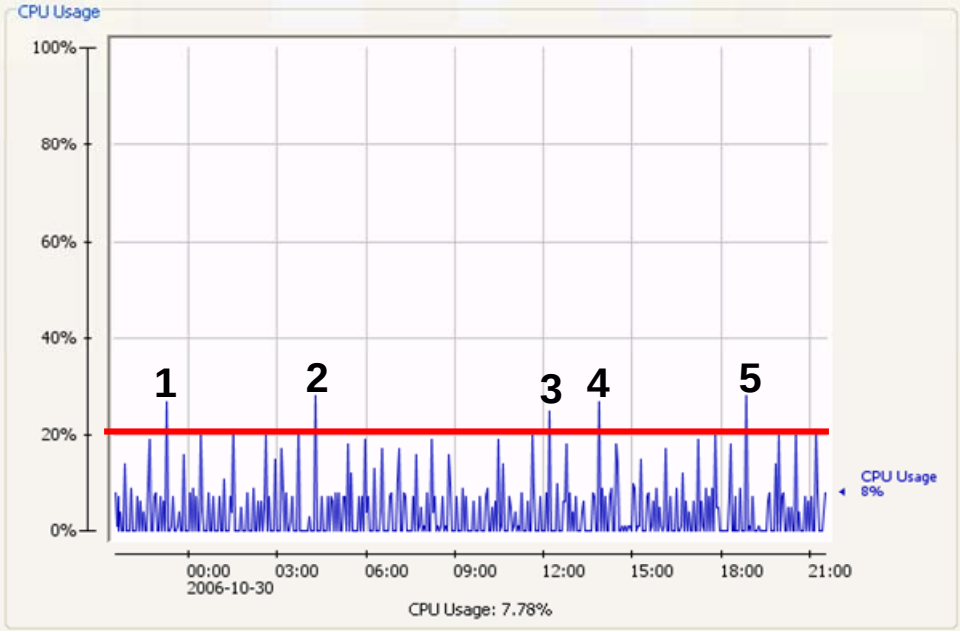

Figure 6.3 Percentage of the hosting node CPU used by the SBLOMARS software package

As far as memory consumption is concerned, Figure 6.4 reveals fluctuations from 1.0Mb to almost 2.0Mb for the same observation period. In Figure 6.4 we are also showing the amount of memory used (1.0MB), committed (memory reserved by the program itself) and maximum amount of memory available.. Keeping in mind that the RAM in the test node is 512 MB, we can say that the SBLOMARS impact is negligible despite the fat that the monitoring agents are non-stopping (they are sensing the corresponding resource without stopping)). A non-stopping

system should handle their memory consumption perfectly. Once a system fails in this activity, there is no opportunity to be used in large-scale Grids.

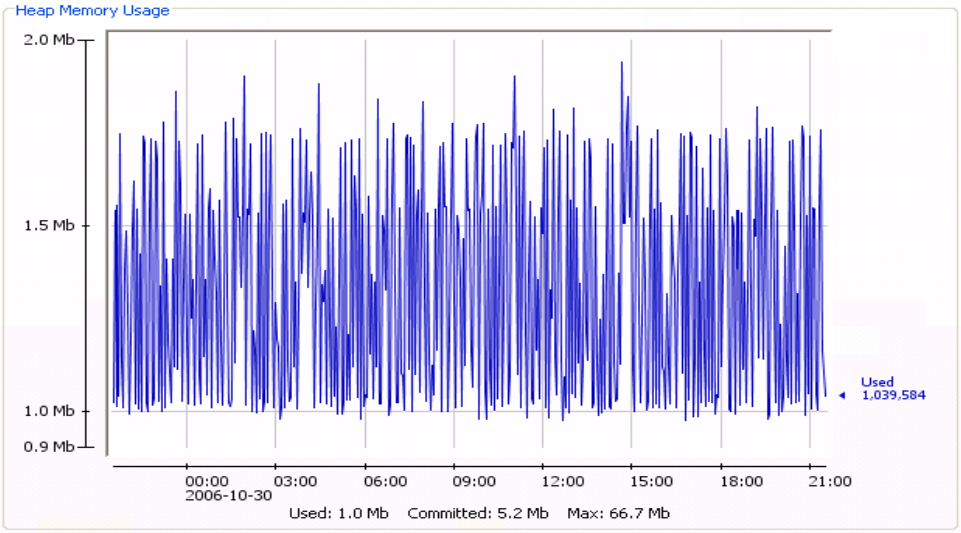

Figure 6.4 Memory usage by the SBLOMARS software package

So far, we have stated that SBLOMARS deploys a set of monitoring agents for each kind of resource to monitor. For instance, a total of five monitoring agents are instantiated in a workstation with two hard disks, one containing two partitions; one bank of memory, and only one CPU.. From a software perspective, each agent instance is a software threat, Therefore, it is important to realize that SBLOMARS is controlling these software threads. Figure 6.5 shows the evolution of the total number of software threads. Misbehaviour would be a constant growth of the number of such threats. Nevertheless, it is clear that we have a constant average trend as many threads appear and disappear from the monitoring system. Therefore we conclude that SBLOMARS will never overload its hosting node capacity.

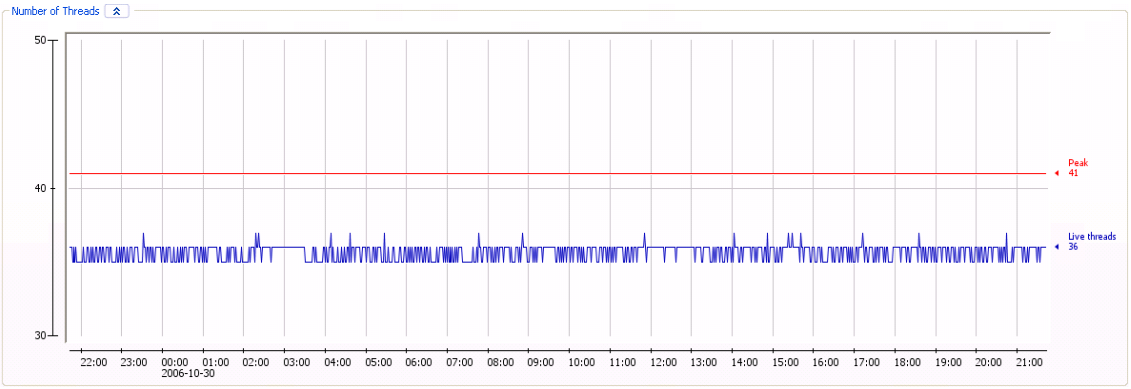

Figure 6.5 SBLOMARS software threads management

In summary, SBLOMARS monitoring agents do not affect hosting system performance despite their continuous resource performance sensing. This statement is based on the fact that SBLOMARS does not consume CPU and memory resources that could affect the normal performance of its host node. In addition, it also efficiently controls the process of creating and removing instances (software threads) from the host node.

6.3.2 SBLOMARS Flexibility Evaluation

Flexibility was defined as the ability to adapt to new resources, resource state, incoming standards and technology. SBLOMARS reaches a high level of flexibility in two ways: The first becomes visible through the implementation of the dynamic software structures, which have been described in Chapter 4. These structures confer flexibility because they automatically handle new resources in the Grid Infrastructure. This is part of the discovery functionality in our design.. The second aspect where the SBLOMARS system grounds its flexibility is through the self-reconfiguration of its monitoring agents polling periods. As already stated, SBLOMARS increases or decreases the interval times between consecutive readings of a given variable based on the state of the monitored devices. This feature in SBLOMARS monitoring system was designed because current monitoring systems for distributed networks such as Ganglia [Mas04] have fixed polling periods.

In order to evaluate the first aspect of the flexibility feature, we deployed SBLOMARS in Athlon, a common storage server unit with Ubuntu Linux 6.10, with three hard disks (Drive C, Drive D, and Drive H) and an external memory disk.. We wanted to demonstrate SBLOMARS capacity to start new sub-monitoring agents (as we described in the Overview section) when new resources are plugged into the SBLOMARS host node.

Figure 6.6 shows the evolution of the capacity used in each disk in this server, for a period of five hours. The experiment consisted in connecting and disconnecting the external disk at different time epochs (at 1:00pm, and at 3:00pm for a period of half an hour). SBLOMARS was able to automatically identify when this device was connected or disconnected and start or stop the corresponding monitoring agent. The importance of this experiment is not just to demonstrate the ability of SBLOMARS monitoring system to perform a new instance when the storage device appears and disappears, but that SBLOMARS keeps the statistical information of this resource in order to be analyzed if this device should be shared into the distributed network. So far, no other resource monitoring system has offered this advantage.

In order to evaluate the second flexibility aspect, we need to remember that SBLOMARS re-configures its polling periods automatically. In fact, SBLOMARS increases or decreases the interval times between consecutive readings of a given variable based on the state of the monitored devices in order to reduce the probability of events misdetection . The device state is fixed arbitrarily by means of two thresholds set on its average usage. If the resource reflects a usage above the higher threshold the polling time is decreased and if the usage is below the lower usage threshold the polling time is increased. The algorithm is restricted to decrease or increase the polling period no more than three consecutive times. This restriction is because reducing more than one third part of the initial polling period could causes some degree of instability of the monitoring systems mainly when the polling period is continually increasing. In other words, the fact of do not limiting the number of times than the algoriothm modifes its polling period could reach infinite values and eventually a memory overload. The code of the algorithm in charge of increasing and decreasing the polling periods is shown in Figure 6.7

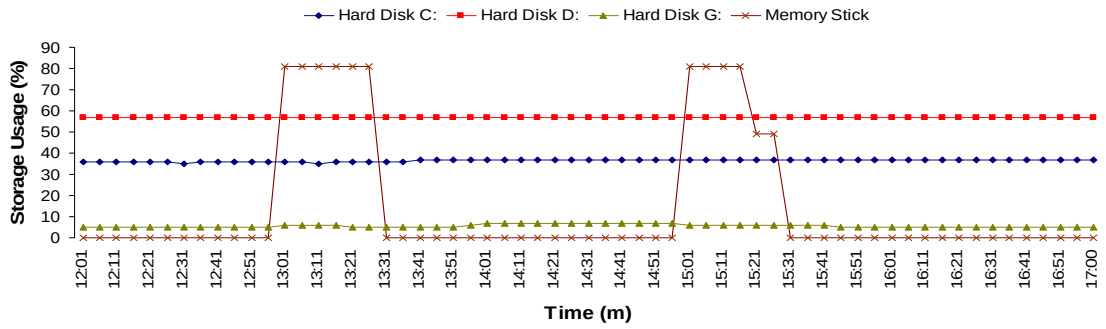

Figure 6.6 Evolution of the used storage capacity over a five hours observation time for a four disks storage system.

.

```
if (averagePercentage > 40) {
alarmThresholdPositive++; }

if (averagePercentage < 20) {
alarmThresholdNegative++; }

if (alarmThresholdPositive >= 1) {
alarmThresholdNegative = 0;
alarmThresholdPositive = 0;
if ( breakingFlagReconfigurationNegative != 0)
     breakingFlagReconfigurationNegative--;
     breakingFlagReconfigurationPositive++;
     if (breakingFlagReconfigurationPositive <= 2) {
       if (_duration > 5) {
       System.out.println("Re-Configuration NEGATIVE");
       reconfiguringFile creatingFile = new reconfiguringFile ();
       int duration = (_duration*50)/100;
String newValues [] = {"timingValuesProcessor.conf",
Integer.toString(duration), Integer.toString(processor_cicles)};
creatingFile.write(newValues); }
}
if (alarmThresholdNegative >= 1) {
alarmThresholdPositive = 0;
alarmThresholdNegative = 0;
//Calling Re-Configuration Method – modifying values from File
if (breakingFlagReconfigurationPositive != 0)
     breakingFlagReconfigurationPositive--;
     breakingFlagReconfigurationNegative++;
     if (breakingFlagReconfigurationNegative <= 2){
       if (_duration < 60) {
       System.out.println("Re-Configuration POSITIVE");
       reconfiguringFile creatingFile = new reconfiguringFile ();
       int duration = _duration*2;
String newValues [] = {"timingValuesProcessor.conf",
Integer.toString(duration), Integer.toString(processor_cicles)};
creatingFile.write(newValues); }
}
```

Figure 6.7 SBLOMARS algorithm for self-adjusting the polling period

Figure 6.8 plots the CPU usage measured by SBLOMARS in one of the nodes from the GRID5000 test-bed (node-20.toulouse.grid5000.fr). This graph was created using a fixed 20 second polling time, thus inhibiting its self-configuration capability. On the other hand, Figure 6.9 shows the result of the same experiment in the same node but with the self-configuration capability activated. Comparing both figures, specifically in the time intervals a), b) and c), we conclude that allowing the self-configuration capability, SBLOMARS is detecting usage events that otherwise would be missed.

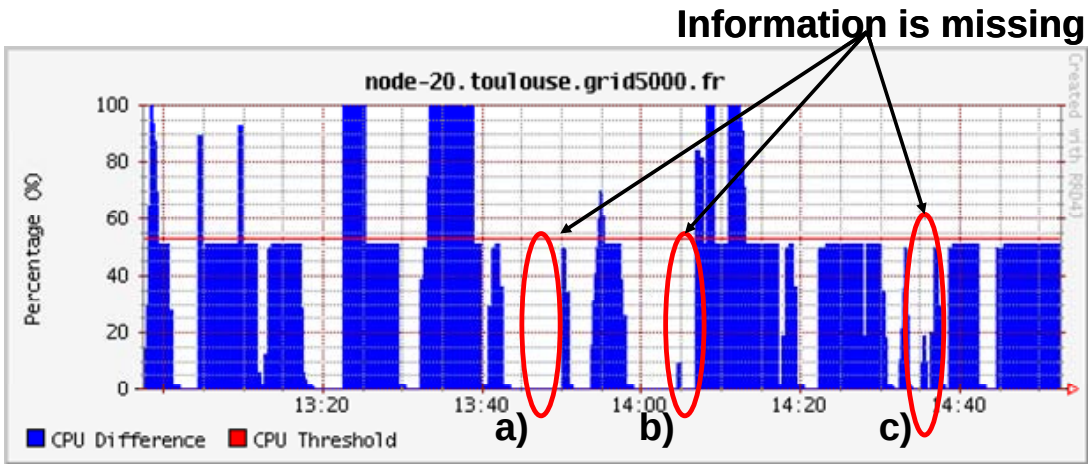

Figure 6.8 CPU capacity usage results with fixed polling times

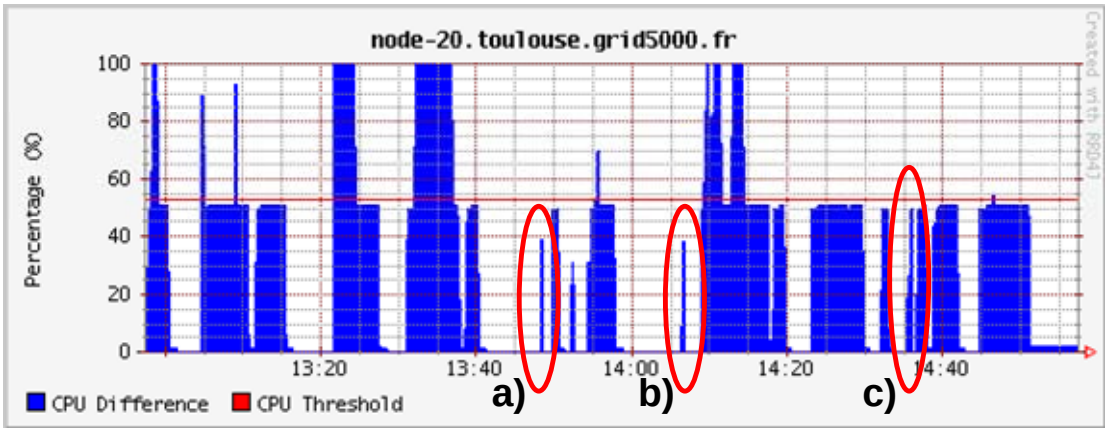

Figure 6.9 CPU capacity usage results with auto-configured polling times

It's clear that reducing the polling periods will result in an increase of data exchange through the network and this may cause overload. Thus, allowing self-reconfiguration we pretend to avoid it and at the same time keep low the misdetection event probability. Nevertheless we have to show that this self-reconfiguration capability is not seriously impacting on the hosting node performance. With this purpose in mind, we did an experiment consisting of varying the polling time through 10, 20, 30, 40, 60, and 120 seconds. The results are shown in Figure 6.10 where we have plotted SBLOMARS CPU usage versus time. Vertical red lines indicate when the re-configuration of the polling time has taken place. When SBLOMARS uses its shortest polling interval, the maximum CPU used by it less than 7% of the total capacity of its host node. This supports the statement that SBLOMARS is not an overload factor when it is

monitoring resources which normally present quite active behaviour, such as CPU or Network Interfaces.

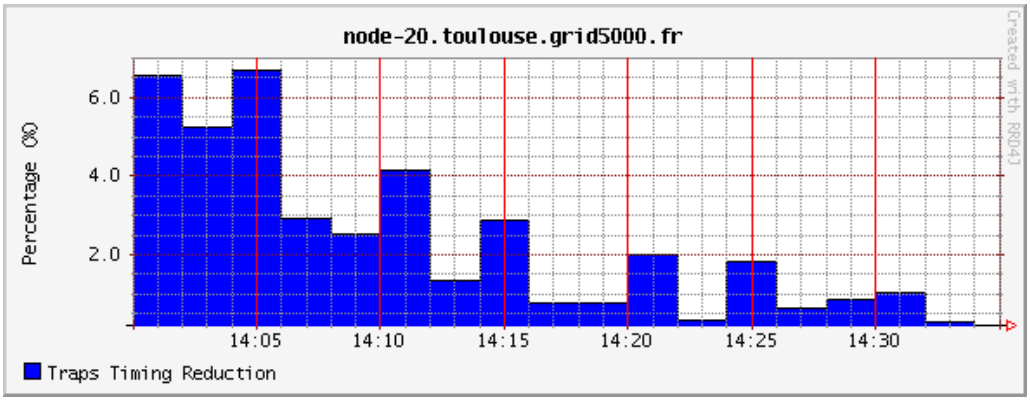

Figure 6.10 SBLOMARS processor overload with polling time self-configuration activated

6.3.3 SBLOMARS Heterogeneity Evaluation

SBLOMARS is capable of monitoring heterogeneous systems. As proof of this assertion we designed an experiment intended to collect resource availability information from systems with three different operating systems: Windows XP, Linux Debian, and Solaris 8. Figure 6.11, shows the CPU behaviour of three nodes (every node is running under the before-mentioned Operating Systems). SBLOMARS was able to show CPU performance of any node regardless of their operating system. The heterogeneity was also demonstrated during the experiments on the Grid5000 test-bed, which is a heterogeneous platform.

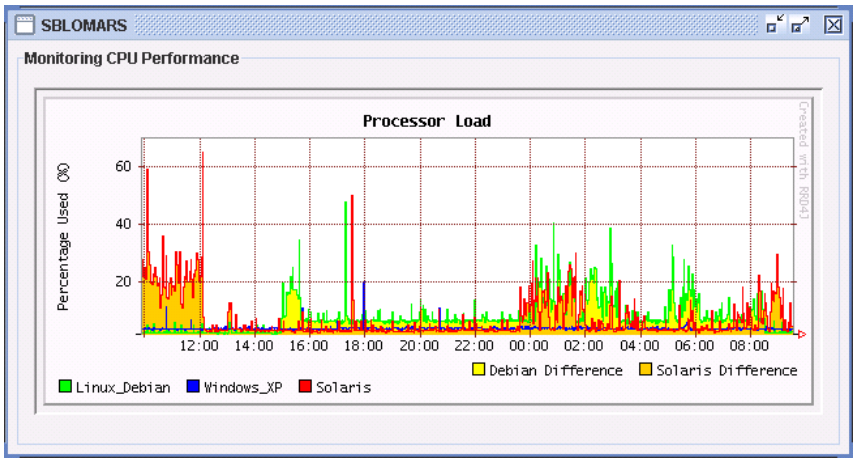

Figure 6.11 SBLOMARS CPU monitoring results of different platforms

6.3.4 SBLOMARS Scalability Evaluation

The scalability evaluation of SBLOMARS monitoring agents was performed in the Grid5000[3] test-bed. In this case we used 180 nodes belonging to this Grid, each one with different architectures. A process generator was used to dispatch processes along the Grid in order to emulate real conditions for all the nodes involved, so as to ensure results approaching real Grid Environments. SBLOMARS was previously installed in all nodes by means of remote scripts that copy, compile, and execute the SBLOMARS code. Connectivity was achieved by means of "SSH" sessions. Figure 6.12 shows the availability results of six out of the 180 intervening nodes along seven hours of test. The colour of the traces in each picture are for the available CPU (blue), memory available (orange), network cards communication rate (yellow), storage available (cyan). A threshold arbitrarily set to 50% is shown as a red line.

From this test we conclude that SBLOMARS gets real-time usability information about any type of resource from 180 nodes without any problem for the duration of the experiment.

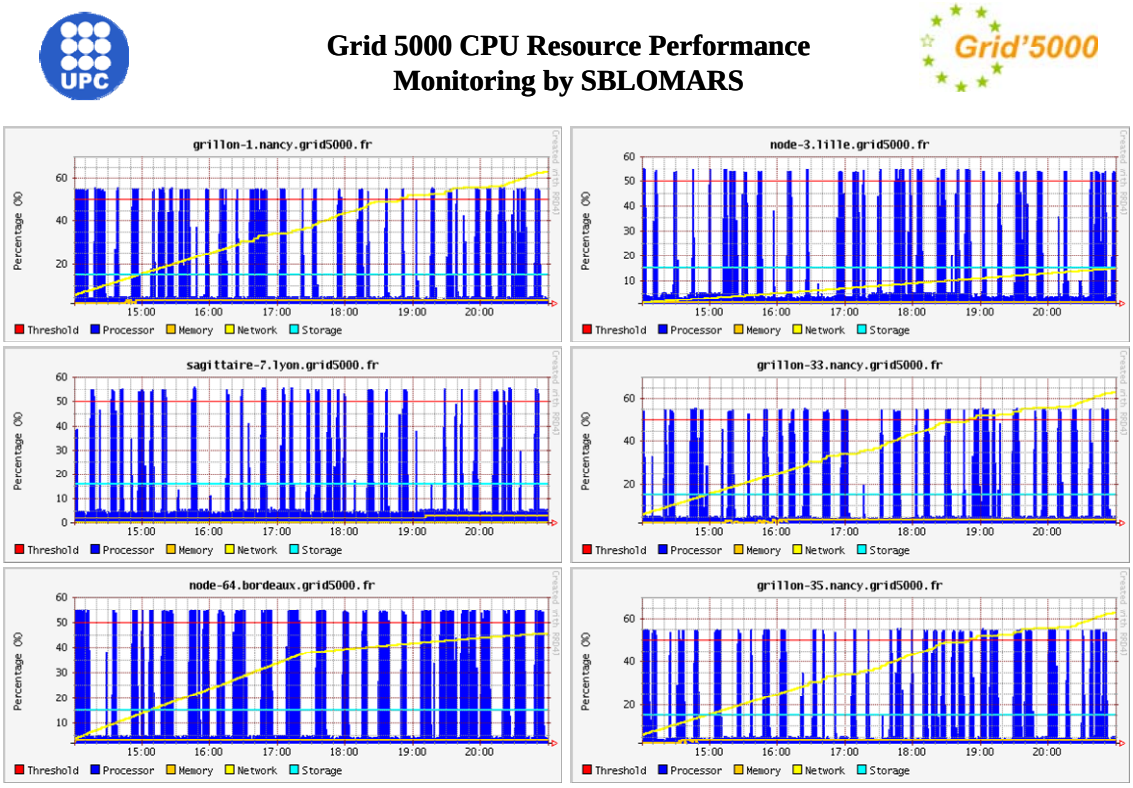

Figure 6.12 Graphical Interface snapshot of the SBLOMARS System in the Grid5000 test-bed

[3]Information about Grid 5000 node architectures can be found in www.grid5000.fr

In addition to the setup of an experiment like this one, it is necessary to carry out three additional phases to get information from the SBLOMARS system. The first one is the ***Configuration*** of the monitoring agents. In this activity each node will receive the parameters used by SBLOMARS to configure its environment. These parameters are the initial polling period, the activation of the flexibility mechanisms, and the number of traps needed to generate a statistical report. The second phase consists of sending the ***Activations*** command to every node where SBLOMARS has been configured. The last phase is to ***Collect*** some resource behaviour information from different nodes. These activities are part of SBLOMARS functionality. Therefore, we have also tested how good the scalability is in each one of these.

Figure 6.13 shows the time required to configure all nodes running SBLOMARS in the Grid5000 test-bed. This is the time taken by the first of the above-mentioned phases. We started with just five nodes and then we increased the number of nodes in increments of five until we reached 115 nodes.[4].

The resulting graph shows that SBLOMARS increases the configuration time in a fair way.. This time is increased steadily because of the network traffic in the test-bed. We cannot control the traffic between clusters which form the Grid5000 and the communication between nodes is not easily controlled. Fortunately, this does not affect our results, as we show in Figure 6.13, Figure 6.14 and Figure 6.15.

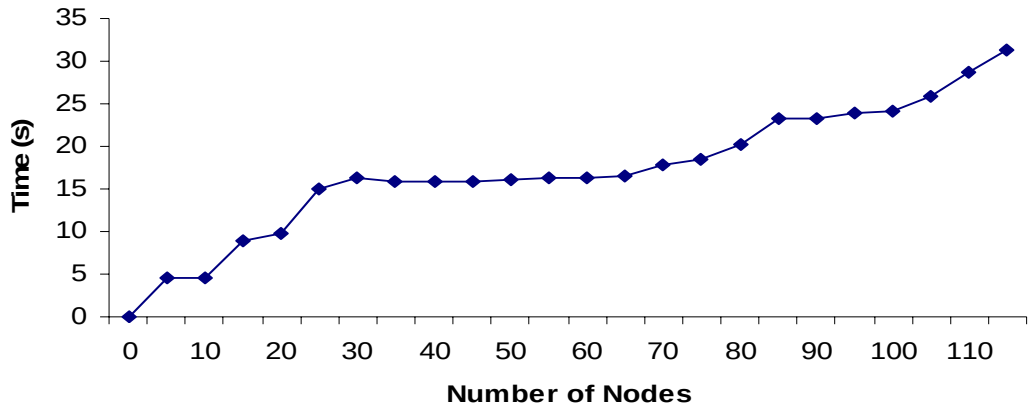

Figure 6.13 SBLOMARS configuration time vs the number of nodes in the Grid5000

[4] Unfortunately, for administrative reasons and scheduling times, in this experiment we were not able to reserve more nodes in Grid5000

We performed the same experiment with regards to the Activation phase, Figure 6.14 shows that the time required by the Activation phase is independent of the number of active nodes.

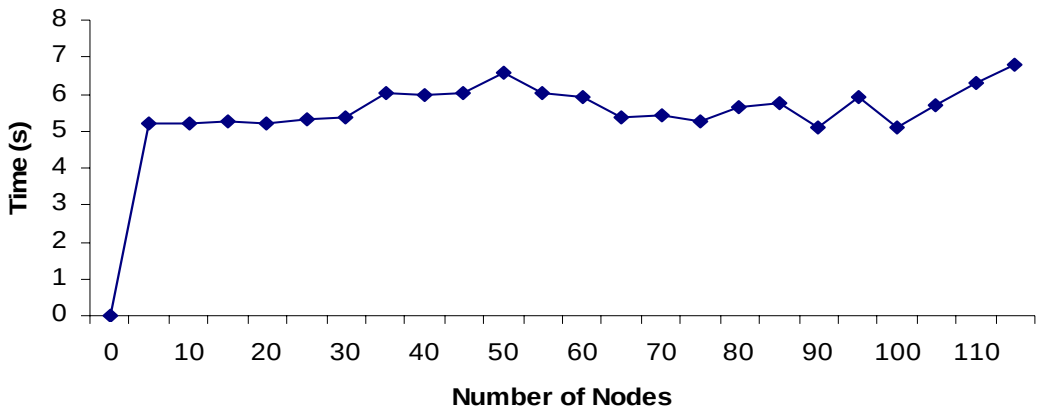

Figure 6.14 SBLOMARS activation time vs the number of nodes in the Grid5000

Finally, we also tested the scalability of SBLOMARS at the Collection phase. with an experiment with the same structure as the previous one. The collection process is between the SBLOMARS agents and the entity requesting resources availability information. The requesting entity could be a user or administrator who wants to know resource behaviour information in certain nodes. Results of Figure 6.15 show an oscillating behaviour. This is attributed to the fact that the Grid5000 test-bed has a high level of networking activity that we could not remove for our experiments.. Nevertheless the trend of the average remains constant regardless of the number of nodes in the experiment.

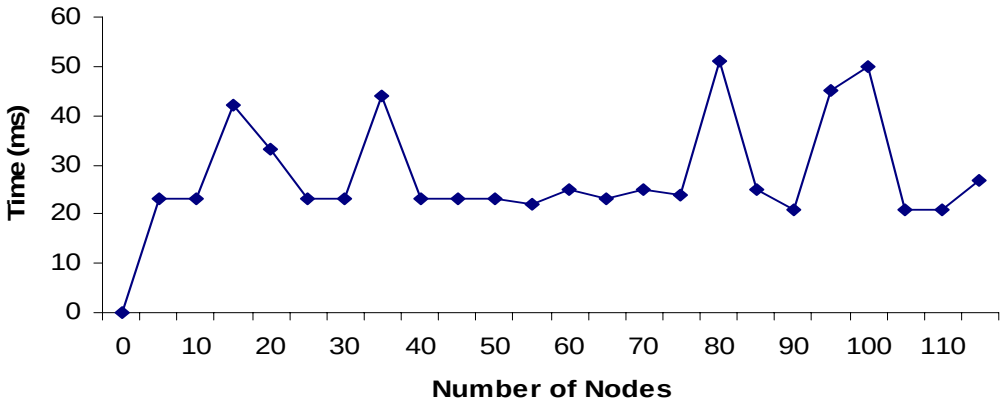

Figure 6.15 SBLOMARS collection time vs the number of nodes in the Grid5000

6.3.5 SBLOMARS Storage Needs Evaluation

In Chapter 4 of this thesis, we have explained that SBLOMARS generates reports about statistical resource availability to be used in the scheduling phase of the Grid Resource Management process. These reports are done in a flexible way, because the number of reports is inversely proportional to the time between polling times. When SBLOMARS is configured, it is necessary to specify the number of values of each variable that have to be collected to generate a statistical report. When this number is small the number of reports will be bigger. For instance, if in one hour SBLOMARS issues a total of 360 polls and a statistical report needs 10 values of a variable to be created, there will be a total of 36 statistical reports, whereas if reports are generated every 2 polls, the total number of statistical reports in one hour will be 180.

These reports are expressed in XML-based documents, which require certain space to be stored. Therefore, it is also important to know the total space used by these reports in a real scenario. In Table 1, we present for each type of resource, the polling time, the total amount of reports generated, and the total space used by these reports in one of the nodes of the Grid5000 test-bed. Some resources are more active than others and there is not necessary to set the same polling period for all of them. These values apply for a time interval of twenty-four hours due to the fact that the SBLOMARS agents automatically clean the memory buffers after this period, rewriting from the oldest to the newest document. This avoids the possibility of filling the system buffer and storage devices with monitoring reports.

TABLE 6.1. STORAGE USED BY THE SBLOMARS DATABASE

Resource	Polling Time (s)	Total # of Reports	Space Used (MB)
Processor	10	8640	3,52
Memory	60	1460	0,576
Network	30	2880	1,143
Storage	300	288	0,357
Software	1800	48	0,212

6.4 BLOMERS EVALUATION

In Chapter 5 we presented the Balanced Load Multi-Constrained Resource Scheduler (BLOMERS) as a resource scheduling system that implements an heuristic approach (Genetic Algorithms) in order to improve the scalability problem and, by making use of real-time and statistical resource availability information generated by SBLOMARS, it schedules jobs in such a way that network activity and resource load remain balanced throughout the Grid.

Heuristic methods do not guarantee an optimization of the scheduling problem. Fortunatelly, there are mechanisms that calculate estimations of how good a heuristic solution is. Analytical Methods [Fis80], Empirical Testing [Law85], and Statistical Inference [Rev95] are among the most well known.

In the following sub-sections we will describe the analytical evaluation of the BLOMERS approach. In particular, the empirical evaluation was done using the Grid 5000 test-bed comparing BLOMERS with the Round-Robin scheduling and Least Average algorithms. The statistical inference and empirical evaluation were left out of the scope of this thesis.

6.4.1 Analytical Evaluation

Evaluating the BLOMERS approach, corresponds to estimating the average fitness of individuals (group of available resources) matching a required schema (conditions for resource load balancing and minimizing makespan).

Be *n(H, k)* the number of individuals in the population *(Pk)* matching schema *H* at generation *k*. If fitness proportional selection is used, and ignoring the effects of crossover and mutation, the expected number of individuals matching *H* at generation *k + 1* is: [Fis80]

$$E(n(H,k+1)) = \sum_{i \in P(k) \cap H} \frac{f(i)}{\overline{f(k)}} \tag{6.1}$$

Where *(Pk)* ∩ *H* denotes the individuals in *Pk* matching *H; f(i)* denotes the fitness of *i,* and *f(k)* denotes the average fitness of the population at time *k*. Expression 6.1 can be rewritten as:

$$E(n(H,k+1)) = \frac{u(H,k)}{\overline{f}(k)} h(H,k) \tag{6.2}$$

$$u(H,k) = \sum_{i \in P(k) \cap H} \frac{f(i)}{n(H,k)} \tag{6.3}$$

The function *u(H, k)* is the average fitness of individuals matching *H* at time *k*. If we denote by *Dc(H)* and *Dm(H)* the probability that an individual matching *H* at generation *k* will be disrupted by crossover or mutation and not match *H* at generation *k + 1*, and assume crossover and mutation to work independently of each other, a lower bound on *E(n(H, k + 1))* is: [Fis80]

$$E(...) \geq \frac{u(H,k)}{\overline{f}(k)} h(H,k)(1 - Dc(H))(1 - Dm(H)) \tag{4.4}$$

Here we ignore the beneficial effects of crossover and mutation. The disruption probabilities *Dc(H)* and *Dm(H)* depend on the details of the operators used, but for the classical choice of one-point crossover, *Dc(H)* will increase with the length of *H*. Assuming the mutation process mutates the individual bits with equal probability, *Dm(H)* will increase with the order of *H* and the number of all possible solutions in *H (combinations possible of H).* Equation (4.4) is known as the schema theorem [Hol75]. More in-depth discussions of the schema theorem, as well as other theoretical approaches to genetic algorithms and evolutionary computation, can be found in [Rev95].

Figure 6.15 presents the effectiveness of the BLOMERS resource selection algorithm versus a random resource selection algorithm based on the above formulae. This effectiveness is given in percent and represents how close to the best one is, as established from the network administrator view. In summary, the BLOMERS approach has better performance than a random search algorithm. Also, we can observe in Figure 6.15 that genetic scheduling improves when the number of resources available is increasing, contrary to the random algorithm whose performance is getting worse.

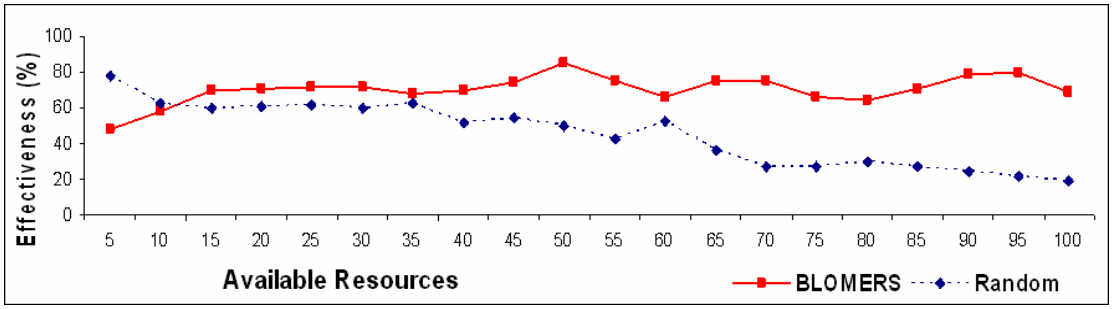

Figure 6.16 BLOMERS vs the Random Selection Algorithm based on analytical evaluation

6.4.2 Performance Evaluation

In order to analyze the BLOMERS performance impact on its hosting node we have performed similar experiments to those effectuated with the SBLOMARS monitoring system. With this in mind, we installed BLOMERS resource scheduler into a Pentium IV workstation with 512MB of RAM memory and Ubuntu Linux 6.10 OS.

The performance results were obtained by means of the same Java Profiler programmer's tool [JAPRO]. In the current experiment, we observe BLOMERS at the resource scheduling phase. The resource scheduler will receive a set of constraints every sixty seconds from the jobs that need to be matched with available computational resources from 120 nodes. In order to simplify this experiment we have avoided the use of SBLOMARS by emulating the information about availability of the resources. This information was previously obtained by SBLOMARS monitoring agents into the Grid5000 text-bed. The whole experiment lasted for two hours.

Figure 6.17 is a snapshot of the CPU used by BLOMERS running the above mentioned experiment for two hours. Average CPU usage is about 6% with peaks rising about 15%. Nevertheless, the impact on other processes running in the same workstation is negligeable because these peaks last for no more than 7 s as illustrated by Figure 6.18. It is clear that BLOMERS resource scheduler does not have priority versus other processes. Therefore, any new process to be executed will not be queued by the CPU broker. Despite of these positive results, BLOMERS will not be executed full-time as a background process with SBLOMARS. In any Virtual Organization there will be running a scheduling system inside a server with a global view of the entire Grid Infrastructure.

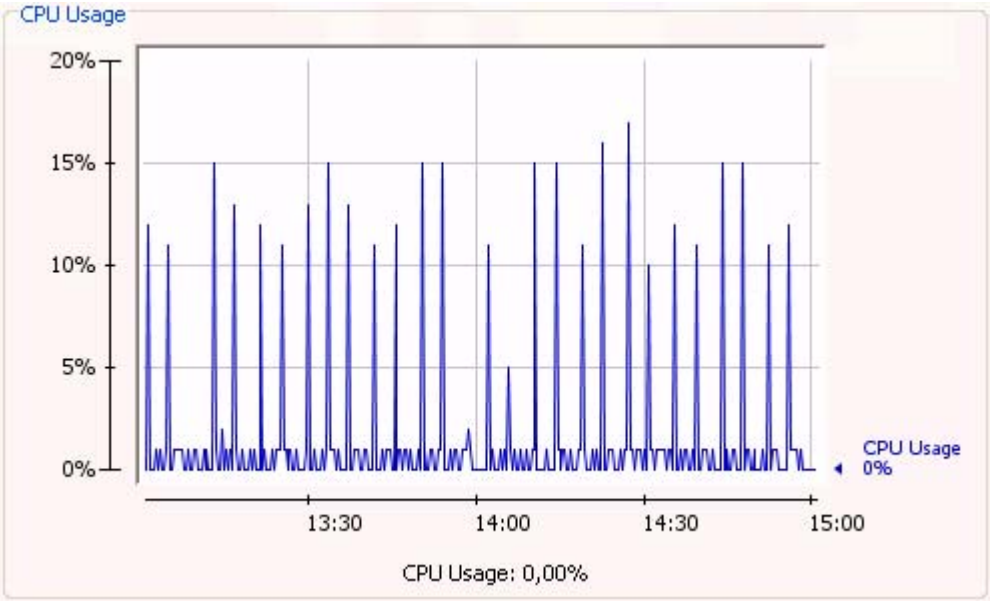

Figure 6.17 Percentage of CPU used by BLOMERS

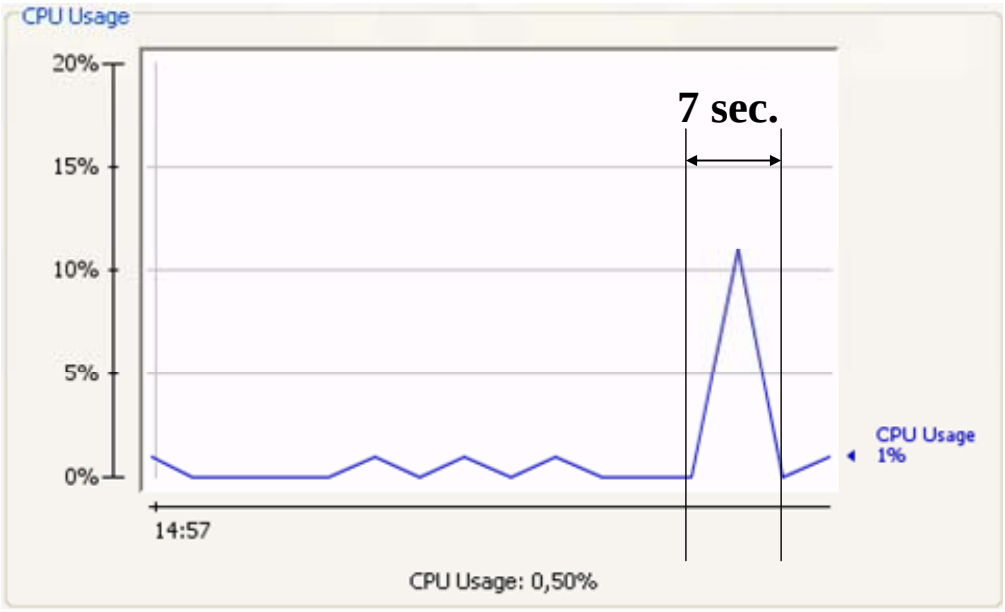

Figure 6.18 CPU used by BLOMERS in 1 minute time span

It is also important to ascertain memory consumption when the scheduling phase is activated. In Figure 6.19 we show the amount of memory that BLOMERS needs to perform its functionality during the same two hours of the previous experiment.

In BLOMERS, memory consumption is higher than SBLOMARS because the new populations (sets of possible solutions) are stored in memory.

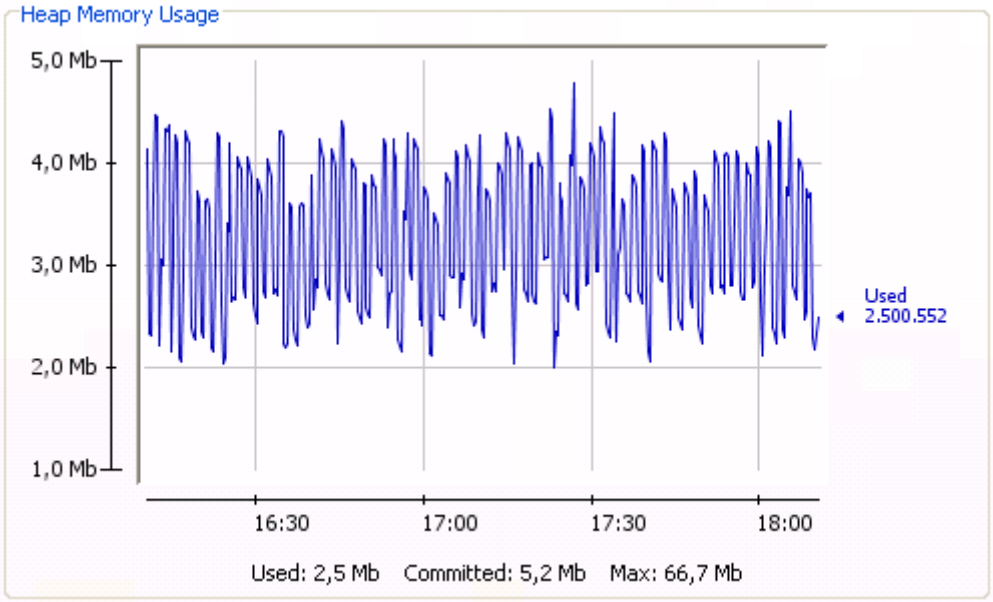

Figure 6.19 Memory used by BLOMERS in its hosting node

The BLOMERS performance evaluation is concluded with the analysis of the number of threads deployed by this system. In Figure 6.20 we show that BLOMERS only deploys 11 or 12 threads to perform its scheduling activities. These relatively low values are obtained because BLOMERS does not perform analysis of the multi-constraint requirements from Grid Services. This analysis is part of the Instances Manager functionality, which is embedded in the Policy-based Grid Resource Management Architecture (PbGRMA).

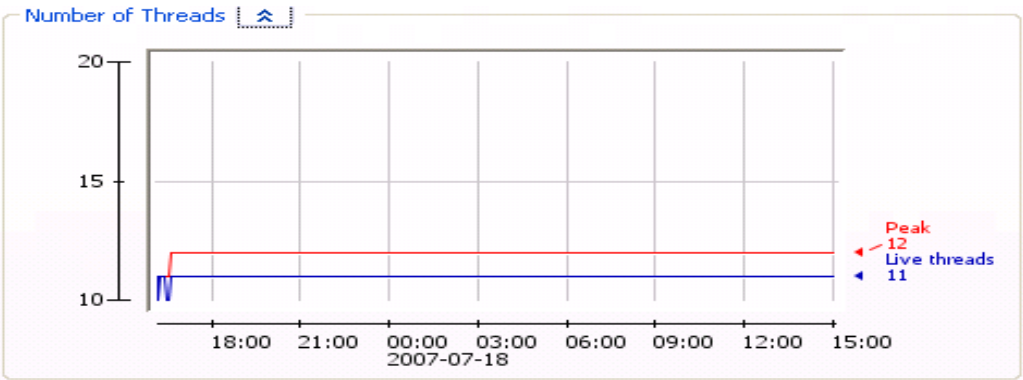

Figure 6.20 Evolution of the BLOMERS software threads

6.5 COMPARISON OF SBLOMARS AND BLOMERS WITH OTHER APPROACHES

We believe that a quantitative performance comparison is unfair for other resource manager approaches; it is ponderous to deploy several resource management systems running the same applications in order to compare them. Due to the differences in the ways our system and the other existing scheduling systems are triggered, quantitative analysis and comparison are not practical. However, we present a qualitative analysis comparing our scheduling approach with other scheduling mechanisms or systems. Therefore, a features comparison is the fairest way to highlight the advantages and disadvantages inherent in these systems.

In Chapters 4 and 5 of this thesis, we have explained that SBLOMARS and BLOMERS are independent, but cooperating to carry out the Grid Resource Management process. Therefore, we need to compare both systems together with current monitoring and scheduling alternatives. In fact, most of the latter have implemented both functionalities (monitoring and scheduling) into the same architectures. Table 2 and Table 3 show this collation with the aim of illustrating the differences between different architectures. We clearly see that SBLOMARS distributed monitoring agents and BLOMERS resource scheduling algorithms are quite competitive, because they offer a fully distributed Grid Resource Management approach. The main facts to highlight for our approach are: Its ability to handle different resource constrains (resource requirement sources), the wide range of computational resources to monitor, its fully distributed architecture, which allows a high level of scalability, and its heuristic implementation in the resource scheduling phase, which increases its ability to minimize the makespan in every service requested.

TABLE 6.2. FEATURES COMPARISON WITH MOST COMMON GRID MONITORING SYSTEMS.

SBLOMARS	MONALISA	GANGLIA	NETLOGGER
ADVANTAGES: • Real Time and Historical Monitoring • Scalability • Reduction of Taffic • Pre-Scheduling • Resource Heterogeneity • Resource Flexibility • Adaptable to External Tools (RRDTool) • Users and	ADVANTAGES: • Real Time and Historical Monitoring Information • Scalability • Adaptable to External Tools (RRDTool, Ganglia) • Adaptable to External	ADVANTAGES: • Real Time Monitoring Information • Multicast Channel • Clusters Heterogeneity • Scalability	ADVANTAGES: • Real Time and Historical Monitoring • Scalability • Resource Heterogeneity • Resource Flexibility • Users Requirements • Applications

Applications Requirements • QoS Requirements	Schedulers (Condor, LSF)		Requirements
DISADVANTAGES: • Mainly oriented to large-scale Grids • SNMP Required • Every resource should be running an Agent	DISADVANTAGES: • Introducing monitoring traffic on the Network • Complex Installation • SNMP Required • Resource Flexibility	DISADVANTAGES: • Non Historical Information • TCP Traffic added • Every resource should be running an Agent	DISADVANTAGES: • Not adaptable to External Tools • Not Pre-scheduling • QoS is not included

TABLE 6.3. FEATURES COMPARISON WITH MOST COMMON GRID SCHEDULERS.

BLOMERS	CONDOR-G	GRIDRM
ADVANTAGES: • Real Time and Historical Monitoring • Scalability • Reduction of Monitoring Traffic • Pre-Scheduling • Resource Heterogeneity • Resource Flexibility • Adaptable to External Tools (RRDTool) • Users and Applications Requirements • QoS Requirements	ADVANTAGES: • Resource Heterogeneity • Resource Flexibility • Users Requirements • Applications Requirements • QoS Requirements (Administrator View) • Globus Adaptability	ADVANTAGES: • Distributed System • Scalability • Fault Tolerance • Security • Resource Heterogeneity • Users Requirements
DISADVANTAGES: • Mainly oriented to large-scale Grids • SNMP Required • Every resource should be running an Agent	DISADVANTAGES: • Not oriented to large-scale Grids • It assumes one entry point into the grid • Multi-domain Issues	DISADVANTAGES: • Not oriented to large-scale Grids • Limited information about resources • Scalability Issues

In spite of the above mentioned difficulties to qualitatively compare different systems in the same environment there is a particular case that for its simplicity we would not dismiss. It is the case of the Round-Robin and Least Average Used resource selection algorithms. Both algorithms can be easily implemented and therefore this is an opportunity for comparison with the BLOMERS approach.

I Due to space limitations we cannot include all sets of results and we present only a sample in Figure 6.21. This experiment reveals the behaviour of the three scheduling algorithms under comparison; namely BLOMERS (blue trace), Round Robin (Yellow trace) and Least Used (Orange trace) in the process of scheduling a system processor.

. The analysis of the complete set of results reveals that BLOMERS is closer to the ideal goal of a scheduler than the other two. In fact, the ideal situation should be when the number of times that this threshold is crossed tends to zero; here we show that BLOMERS is the best. BLOMERS, beside of reducing the makespan for the entire scheduling process as other schedulers do, goes a step further because the resource load all over the Grid is much better than with other algorithms

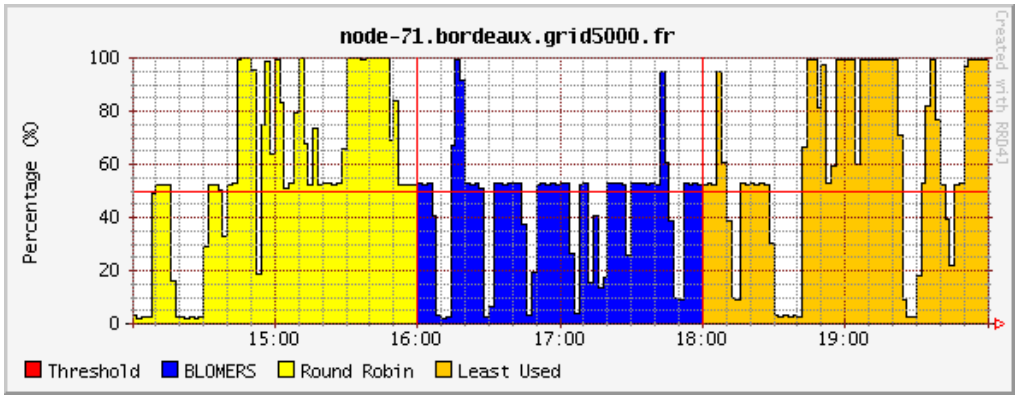

Figure 6.21 BLOMERS Vs the Round-robin and the Least Average Used scheduling algorithms

6.6 PbGRMA Evaluation

The evaluation of the Policy-based Grid Resource Management Architecture (PbGRMA) was done in two phases. The first one was performed when the necessary components were adapted and implemented to improve our initial PbMA [Sal03] at the Grid Computing Domain. This process, detailed in [Maga05] and [Maga07a], can be considered an evaluation of the system in isolation of the SBLOMARS and BLOMERS systems. The second phase was performed on the Grid 5000 test-bed. In this phase the SBLOMARS and BLOMERS systems were used as well.

6.6.1 PbGRMA Performance Evaluation in Isolation

To evaluate the architecture in isolation, we used a set of heterogeneous nodes (i.e. Intel® and AMD®) with different operating platforms (Windows 2000 and Linux Fedora 4), as well as different amounts of resources to share, such as our Grid Infrastructure. The only homogeneous feature of these nodes is that all of them have Globus Toolkit 2.4 installed, and they have been signed by the same Certificate Authority (CA), our own CA, and not the standard Globus-CA, for security reasons. A random process generator was used to dispatch processes to the network nodes so that we could emulate normal “working day” conditions for all the nodes involved, to ensure results resembling real Grid environments. The entire architecture was programmed in Java platform. The components of the PbGRMA were CORBA objects. Policies were expressed in XML, and the interfaces were implemented with the standard Interface Definition Language.

The Grid Service selected to be distributed along the Grid environment was the Newton's Method [Cha00], a generalized process to find an accurate root of the equation f (x) = 0. This method has many physical and astronomical applications and was selected because of the number of the algorithm’s iterations is considerable (therefore the amount of processing resources needed is really significant) and also because this method is quite general, which means that performing these experiments with other Grid Services would generate similar results.

The following experiments intents to show how effective the PbGRMA is when an application is running in only one node instead of being distributed in the maximum amount of nodes. The distribution criteria for this application were to match the maximum amount of parts that the requested Grid Service is able to be distributed with the amount of existing nodes in the Grid Infrastruture. This information was obtained from the WS-DL and WS-Resource Properties documents that we are showing in Figures 6.23 and 6.24 respectively.

The first results are shown in Figure 6.22. In this experiment we plotted the percentage of resources (processor, memory, and storage) used by Newton's application during five sequences, with around one thousand iterations for each one of them (i.e. the same application will be executed five times in order to obtain a more precise result). It is clear that during the application processing time, the resources are at their maximum capability and any other process will be queued and executed after completing the equation. In this experiment the PbGRMA is contacting neither SBLOMARS nor BLOMERS to perform these operations. Therefore, it does not have visibility of the resources availability; we just wanted to show that PbGRMA is able of distributing OGSA-based Grid Services in a fixed configuration network like the one used in these experiments.

In this example, the total time is around seventy seconds (70,000ms), whilst in some astronomic applications it could reach days in simulation.

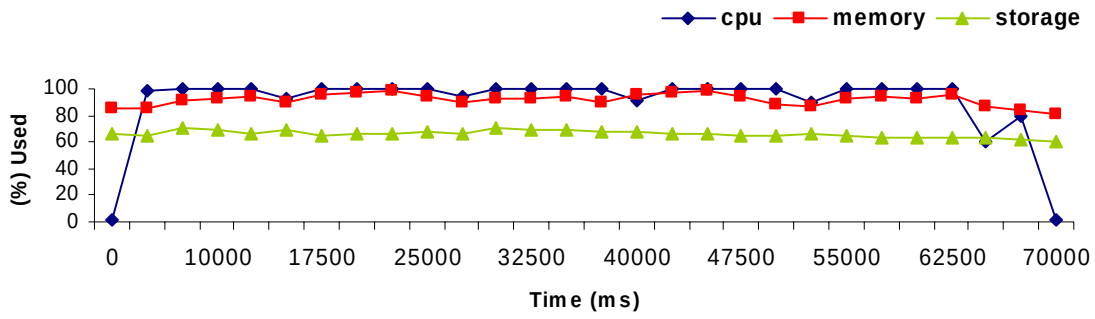

Figure 6.22 Resources' performance with a best effort management policy.

The following test in our evaluation process consists of the insertion of a QoS Policy in the above-described scenario. In this test, the policy demands the maximization of resource exploitation in a minimal amount of time (QoS level policy). To obtain this level of efficiency, the PbGRMA used different nodes for each application sequence, with a maximum of five nodes (Figure 6.24). These service requirements were obtained from both Web Services Description Language and the Web Services Resource Properties documents, presented in Figure 6.23 and Figure 6.24 respectively.

```
<!-- ========== WSDL Interface for Newton's Method Application  ============ -->
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://nmg.upc.es/Newton'sMethodExample" ...>
  <wsdl:types>
     <xsd:schema
       <xsd: import
     targetNamespace="http://nmg.upc.es/Newton'sMethodExample_Properties" ...
       <xsd:attribute name="ResourceProperties" type="xsd:Newton's Method"/>
       ...
     </xsd:schema>
  </wsdl:types>
```

Figure 6.23 The Web Service Description Language interface for the Newton's method

```
<!-- == WS-Resource Properties Document for Newton's Method Application  == -->
<wsdl:portType name="Newton's Method"
     wsrp:ResourceProperties= "intf:GenericMethodProperties">
     <xsd:sequence>
        <xsd:element maxDistribution="5" minDistribution="1" name=" " …
        <xsd:element amountMinMemory="20" amountMaxMemory="250" name=" "...
        … "wsa:EndpointReferenceType"/>
</wsdl:portType>
```

Figure 6.24 The Web Service Resource Properties Document for the Newton's method

The Grid nodes continue their normal activity until the system detects that the percentages of used resources are at low levels in some of them. At this moment, their resources can be shared with any client within the Grid Infrastructure. The PbGRMA distributes the application to the set of selected nodes and remotely executes the applications. Figure 6.25 plots the resources monitoring activity in one of the selected nodes. In order to compare the new processing time versus the previous graph, we have selected the same node. The analysis of this graph illustrates the benefits of our approach because the system offers substantial savings on time and used resources. Moreover, this graph shows the time needed by the server to gather the results provided by the other selected servers and to obtain the final result.

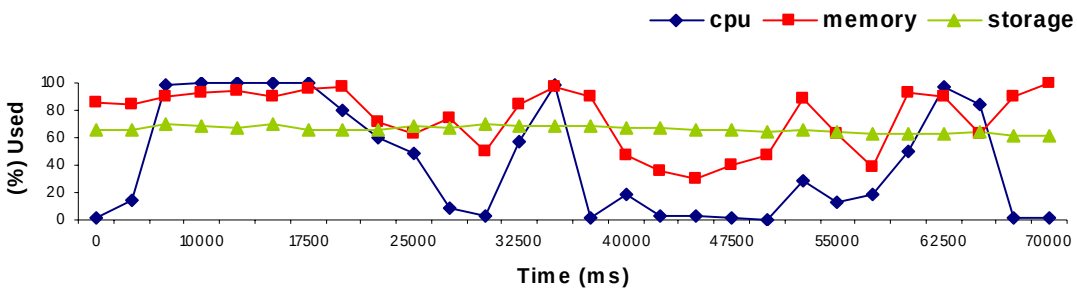

Figure 6.25 Resources' performance with Golden QoS management policy

6.6.2 PbGRMA Performance Evaluation in Grid5000

To evaluate the full PbGRMA, we have used 74 nodes belonging to the Grid5000 test-bed. They were configured following the workflow presented in section 6.2 of this chapter.. We used the same random process generator to dispatch processes along the Grid. The application selected was also the Newton's Method implementation, but with extra set-up parameters, which make the application heterogeneous in terms of CPU and Memory requirements. The reason behind this choice was to guarantee that a large number of resources would be needed, and that a significant load would be placed on the Grid, based on the fact that the algorithm used in this process requires a considerable number of iterations. The entire experiment lasted for 6hrs. Each scheduling algorithm worked for 120 minutes (2hrs), receiving 30 jobs every 60 seconds. In Figure 6.26 we show the CPU percentage used by six out of the 74 nodes during the experiment, with three different scheduling algorithms as follows:

- **BLOMERS:** This algorithm is described in detail in Chapter 5. It is based on a Genetic Algorithm and SBLOMARS monitoring agents. This algorithm selects the nodes whose processor threshold is not more than 50%.

- **Round-robin:** This algorithm schedules every job received to the next available node from a list of nodes available. Once the list has been completed, this approach sends the next job to the first node of the list, and goes on. It then generates several cycles per scheduling.

- **Least Used**: This algorithm schedules based on the average of the least used node. Based on the statistics generated per SBLOMARS monitoring agents it selects the least used on average and sends the next job received

We have argued during the development of this research that SBLOMARS monitoring agents and BLOMERS resource scheduler working together could improve scheduling times (makespan) and load-balancing on the Grid Infrastructure. We have compared how good the load-balancing feature in our approach is against other scheduling algorithms. The processor behaviour of six nodes from the Grid5000 platform is plotted in Figure 6.26, as a function of the scheduling algorithm used (remaining experiment graphs can be consulted in [Maga06]). Vertical red lines represent the border between each scheduling algorithm, and the horizontal red line shows the threshold used for the BLOMERS selection policy.

The blue area represents the performance of the CPU when BLOMERS scheduling systems is being used by the PbGRMA. Considering that, in an ideal case, the number of times the threshold is crossed should be zero, it is apparent that BLOMERS is the scheduler closest to this goal. On the contrary, the round-robin and least average used algorithms cross in many more times the threshold of the 50% of resources used. It is clear that BLOMERS is a very effective load balancing scheduling algorithm.

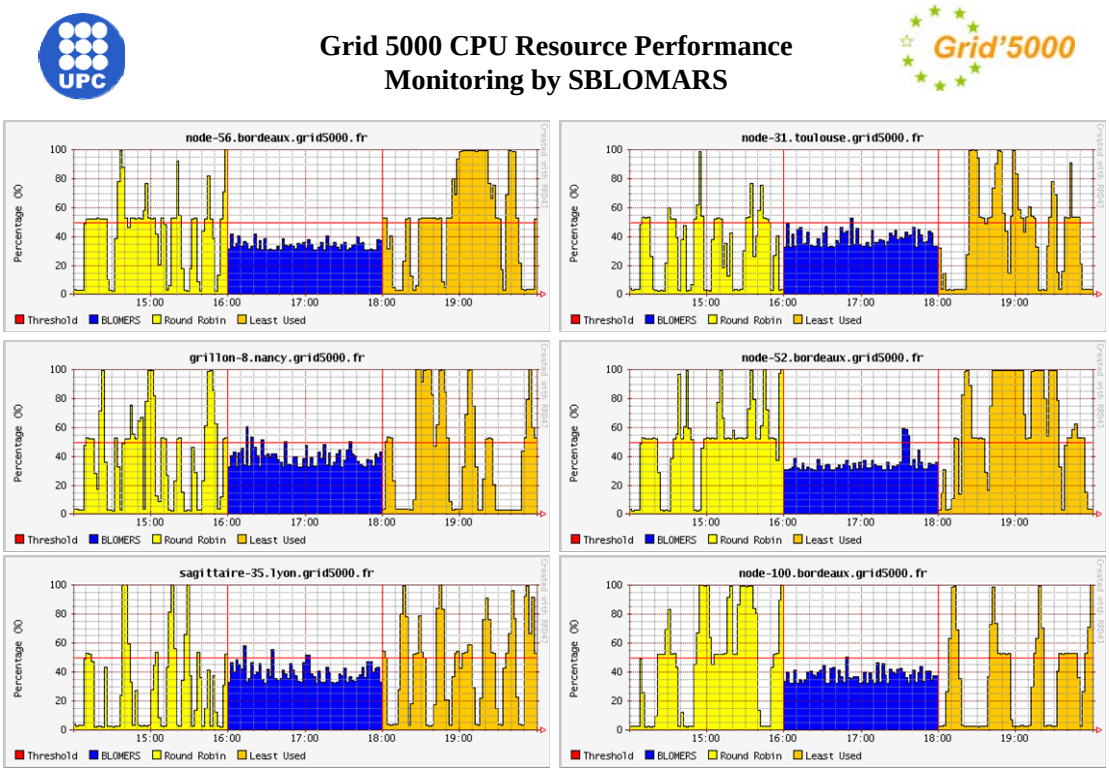

Figure 6.26 SBLOMARS Graphical Interface snapshot of the CPU usage in the Grid5000 test-bed with the full GRM process managed by the PbGRMA.

Alternatively, we measured the time required by the Grid Resource Scheduling (GRS) phase in an incremental scenario. We have one hundred and eighty nodes as a Grid Infrastructure. We started the experiment with just five nodes, and we increased, in increments of five, the number of nodes in the experiment. At the end of the day, the makespan required in our architecture is relatively constant. This is a guarantee that the scalability of the architecture is preserved. In Figure 6.27 illustrates the before-mentioned results.

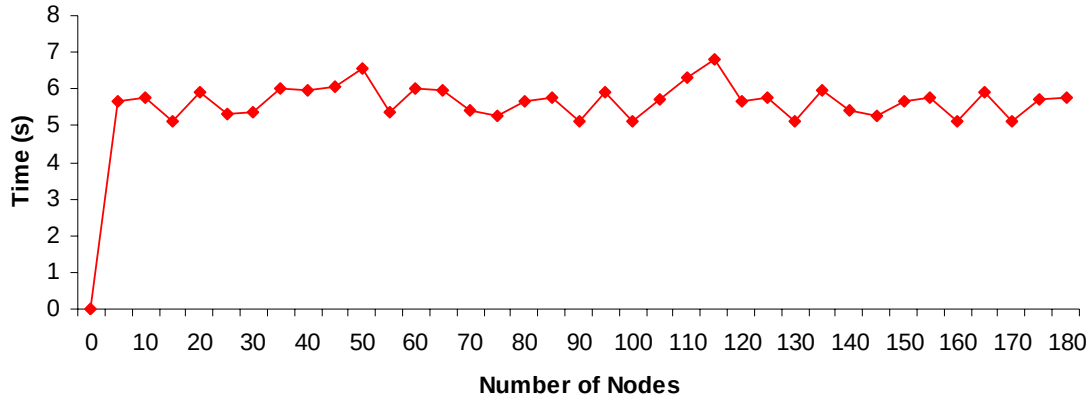

Figure 6.27 Time required to match available resources with required constraints by the Blomers system vs the number of nodes in the Grid5000 test-bed

Having shown the good load-balancing and scalability properties of our approach, we now proceed to investigate the impact of the management system. In these experiments we will show the reliability and performance of the Allocation and Activation (JAA) phase in our proposed Grid Resource Management process.

In this experiment BLOMERS performs the scheduling activities. The SBLOMARS is running as usual on each node forming the Grid Infrastructure for this experiment. This reserved Grid Infrastructure involves 180 nodes from four different clusters: Lyon, Lille, Bordeaux and Nancy. As we have highlighted before, these clusters are heterogeneous. The full experiment was four hours long. We constantly requested a variable number of Grid Services (newton's application), which were split up in small jobs. In our system there is not a component in charge of splitting Grid Services into smaller jobs. We are considering as new Grid Services every application that is going to be run over the Grid Infrastructure. Grid Services were requested at the rate of fifty per minute. In each Service, the QoS level was fixed and the experiment was run in four intervals, each one lasting for one hour and each one with a specific QoS policy.

In Figure 6.28 we show the average processor load results for the total of 180 nodes (green trace)..The amount of CPU used in average for the 180 nodes involved in this experiment was graphed in four segments of one hour each one of them. During every hour a different QoS policy was enforced staring from Diamond (blue), Golden (black), Silver (Orange) and Bronze (pink). Whereas the Diamond QoS Policy exhibits the best performance for the overall Grid, performance at the Bronze policy level is marginal. Obviously, the computational cost increases for higher QoS level policies, but it seldom crosses the threshold shown in the graph.

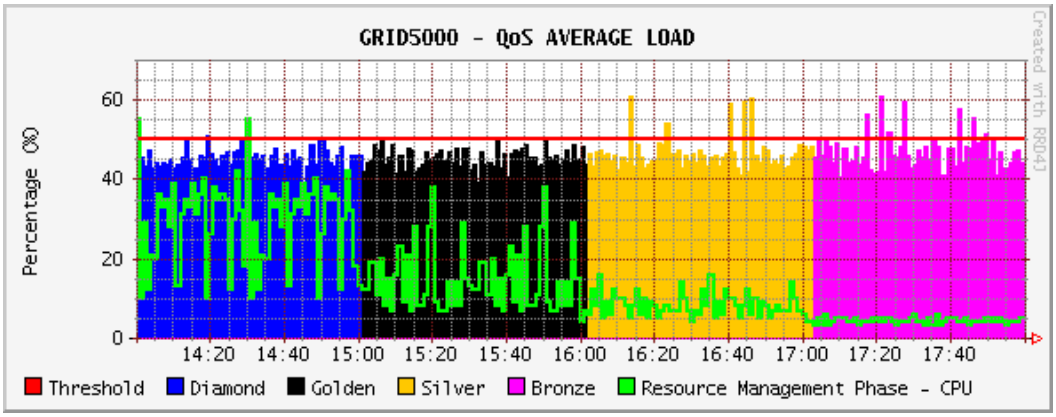

Figure 6.28 The CPU average used by the four QoS policies Vs. the amount of CPU consumed by every QoS policy level

Finally, Table 3 shows the elapsed averaged times measured during the whole Grid Resource Management process. In order to measure these elapsed times we have performed a separate experiment that consisted in the completion of the four phases of the GRM process; namely, Grid Services Management, Grid Resources Discovery and Monitoring, Grid Resource Scheduling, and Jobs Allocation and Activation. The Grid Services Management phase, where the Domain-Level policy is created, is just 750ms which is very short for the full phase. This interval is due to the parsing time needed, because Grid Services are described in standard XML format [XML]. The time consumed by the Grid Resource Discovery and Monitoring (GRDM) phase is not applicable because the time spent between SBLOMARS and BLOMERS communication-flows is part of the Grid Resource Scheduling (GRS).The GRS phase is completed in five seconds, considering the amount of jobs received that were around 36000 per hour and the number of nodes considered (180) in the Grid 5000 test-bed we clearly show that this phase is highly competitve. The Jobs Allocation and Activation (JAA) phase took almost the double of the GRS because the network conditions in the test-bed. Wwe did not considered networking metrics as a part of our experiment. We can improve this times integrating networking metrics as hard constraints for the Grid Services. We show with this times that no matter the size of the Grid Infrastructure, the full deployment and management of Grid Services will remain within these intervals. It was also demonstrated by the experiments shown in sections 6.3.4. and 6.4.1. The scalability of the the entire architecture is also shown and guaranteed by these results. The driving force to improve this architecture will be the possibility of reducing these times, which in turn are directly proportional to the efficiency of the Policy-based Grid Resource Management Architecture.

The times presented in Table 6.3 are high competitive because large-scale infrastructures like Grid5000 normally deploy Grid Services in longer times that the ones obtained by our general approach [Gri5000].

TABLE 6.4. AVERAGE CONSUMING TIMES BY GRID RESOURCE MANAGEMENT PROCESS

Actions	Timing
Grid Services Management (GSM)	750 ms
Grid Resource Discovery and Monitoring (GRDM)	N/A
Grid Resource Scheduling (GRS)	5525 ms
Jobs Allocation and Activation (JAA)	10125 ms

6.7 CONCLUSIONS

In this Chapter we have summarized the experiments done to analyze the three systems forming our proposal for Grid Resource Management process.

A set of experiments were aimed to evaluate the performance, flexibility, heterogeneity, and scalability of our SNMP-based Balanced Load Monitoring agents for Resource Scheduling in Large-scale Grids (SBLOMARS). Specifically, it was very important to demonstrate that the SBLOMARS approach is heterogeneous and flexible, because these two features make it truly novel compared to current monitoring systems for distributed systems and Grids. Moreover, it was essential in our research to deploy our SBLOMARS monitoring agents in a real large-scale scenario, such as Grid5000.

Results of the above tests reveal that SBLOMARS shows a great advantage in front of its competitors in terms of heterogeneity because it is able to monitor heterogeneous operating platforms (Linux, Windows, Solaris, and Macintosh). It also performs very well no matter the number of instances (new monitoring agents) that it has to deploy. On the other hand, SBLOMARS shows a powerful capacity to perform monitoring activities regardless of the architecture or complexity of the nodes forming the network. This was demonstrated by running SBLOMARS for long intervals of time, in at least one-hundred and eighty nodes from seven of the nine clusters forming the Grid5000 test-bed. Finally, SBLOMERS scales very well because in spite of the increase of the number nodes in the Grid, the time to trigger the monitoring agents and the time to collect their information remains quite stable.

Different sets of experiments were conducted to analyze our Balanced-load Multi-Constrained Resource Scheduler (BLOMERS). This heuristic approach is based on a Genetic Algorithm, as we have explained in detail in Chapter 5 of this thesis. Following the most common mechanism to evaluate an heuristic algorithm, we have estimated the reliability in terms of makespan of the BLOMERS resource scheduler. We have also evaluated the performance of the algorithm running some scheduling jobs for a long period of time. Finally, we have deployed our algorithm on the Grid5000 test-bed to perform experiments regarding makespan and load-balancing. We obtained very good results because BLOMERS keeps a very good level of load-balancing along the Grid Infrastructure compared at least with the Round-Robin and the Least Average Used algorithms.

In this chapter we have also included an evaluation of the Policy-based Grid Resource Management Architecture in two steps. The first was with an isolated system, and the second showed it with working together with SBLOMARS and BLOMERS . In the evaluation of the architecture in isolation, we demonstrated its compatibility with external components or middleware. We have interfaced this architecture with Globus ToolKit 2.4 and performed several

experiments to distributed Grid Services, which are described through WSDL and WS Resource Properties documents as a standard workflow implied by the Open Grid Forum (OGF).

Although our PbGRMA is focused on Grids where a wide range of nodes will offer small amounts of resources, the solution is not only limited to this domain of users/clients. Results, presented in the last experiments of this chapter show the advantages of our architecture. In fact, the reduction in processing time is around the 57%, and the percentage of resources used per second in all the Grid Infrastructure is much greater than in other approaches. Therefore our environment is more efficient in terms of resources used. We claim that this is a feasible solution for small business and industrial users wishing an automatic and self-managing effective access to massive amounts of computing, network, and storage resources, reducing procurement, deployment, maintenance, and operational cost.

The second step of the evaluation process for the PbGRMA was performed in order to evaluate the reliability of the whole architecture. This means that the three components were deployed in the Grid5000 test-bed. Based on these experiments, it is clear that our approach is a reliable architecture with good scalability and heterogeneity. In addition we have proved other of its relevant capabilities like QoS constrained deployment and management of Grid Services without overloading the management instances and that it is a standard approach, due to the fact that the OGSA standard was taken into account in our design and implementation.

This chapter also helps to explain how it is possible to use Grid5000 test-bed. We have mentioned several times that one of the most important activities in our research was to evaluate our architecture in a real, large-scale Grid.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 INTRODUCTION

Grid Resource Management (GRM) is regarded as a vital component of the success of the concepts behind Grid Computing because it is intended to efficiently manage the coordination and sharing of multiple computational and networking resources. But GRM faces several challenges that make the implementation of practical management systems a very difficult problem. Moreover, GRM systems must fulfil strict functional requirements from heterogeneous, and sometimes conflicting, domains (e.g., user applications, and network domains) [Nab04]. In addition, GRM systems must adhere to non-functional requirements that are also challenging, such as reliability and efficiency, in terms of time consumption and load balancing of the resources shared by the Grid Infrastructure.

In this thesis we have highlighted that current GRM systems research is focused mostly on solving either one or the other of the two main problems at hand. The first is concerned with evenly balancing resource loads throughout a Grid. In this area, researchers have proposed a range of technologies with varying degrees of success [Kra02]. The second problem involves shortening the amount of time, from start to end, needed for completing a set of. Up to now, a comprehensive solution addressing both of these problems proved elusive; In other words, a load-balanced solution with a minimal makespan did not yet exist. The method we presented in this thesis provides such a solution. Our approach involves splitting and distributing the resource management process into four main phases: Grid Services Management (GSM), Grid Resource Discovering and Monitoring (GRDM), Grid Resource Scheduling (GRS), and Jobs Allocation and Activation (JAA). These four phases are materialized in framework composed of three main independent but cooperating systems to perform the Grid Resource Management (GRM) process.

The GSM and JAA phases are supported by means of a Policy-based Grid Resource Management Architecture (PbGRMA) [Maga07a]. This architecture is able to consider service needs arising from diverse sources during the deployment and management of Grid Services, such as customer requirements, applications, and network conditions. The synergy obtained through these components allows Grid administrators to exploit the available resources with predetermined levels of Quality of Service (QoS), reducing computational costs and makespan in resource scheduling, while ensuring that the resource load is balanced throughout the Grid. The GRDM phase is supported by the SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS) [Maga07b], in which network and computational resources are monitored by distributed agents. This allows the design of a flexible, heterogeneous, and scalable monitoring system. The GRS phase is based on the Balanced Load Multi-Constrainted Resource Scheduler (BLOMERS) [Maga07c]. This heuristic scheduler represents an alternative for solving the inherent NP-hard problem for resource scheduling in large-scale distributed networks.

The remainder of the chapter is structured as follows: Section 2 outlines a review of the contributions presented in this thesis. In this section we also highlight some drawbacks of the three main contributions of this research work; namely PbGRMA, SBLOMARS and BLOMERS. Finally, Section 7.3 presents future work and also shows some new research that could be exploited as a continuation of this work.

7.2 REVIEW OF CONTRIBUTIONS

In this section we will describe the main contributions presented in this thesis.

7.2.1 Policy-based Grid Resource Management Architecture (PbGRMA)

The PbGRMA is a management solution conceived with a granularity of two levels. The Grid Domain Management System (GDMS) and the Grid Node Management System (GNMS). In the context of this thesis, we have understood Virtual Organization (VO) as a Grid Domain level. Therefore, it will be necessary to deploy one PbGRMA per VO existing in the full Grid Infrastructure. System scalability is ensured thanks to this two level management approach..

The introduction of a complex system such as a Policy-based system in the Grid Computing area was to simplify the current communication process between the three main actors intervening at the deployment process of Grid Services. These actors are Grid Infrastructure Providers, Grid Services Customers, and Grid Services Repositories. A Policy-based solution perfectly fits the fussy requirements of the Grid Services deployment workflow. This is due to the flexibility and scalability properties exhibited by policy-based management systems. PbGRMA offers an excellent portal for the deployment of high-level policies. The whole architecture handles different levels of QoS with excellent performance.

One of the most important aspects of our solution is its ability to self-extend many of the management capabilities exhibited by management systems. In fact, the proposed PbGRMA makes autonomous decisions to extend Domain-level and Node-level components such as Policy Decision Points and Policy Enforcement Points. The PDP components are also autonomously extended by means of appropriate Action and Condition Interpreters. This solution offers a high level or granularity, thus making it more scalable.. Moreover, the self-configuration capability of the proposed architecture to deploy or remove new Management Instances improves the efficiency of the management system in terms of computational resource usage.

This Policy-based architecture offers the perfect solution to add as many service management requirements as needed. Our solution admits up to three main sources of such requirements. Namely, user quality of service requirements, Grid Services specifications based on OGSA and WS-RF standards, and Grid resources availability to maintain certain load balance along the Grid Infrastructure. This has never been accomplished before.

On the other hand, the PbGRMA is also in charge of executing the last phase of the GRM process presented in this thesis; namely the JAA, fitting the needs dictated by heterogeneous architectures to allocate and activate Grid Services.

7.2.2 SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS)

The Resource Monitoring and Discovery phase is aimed at determining which resources are available to be assigned to execute a specific job, application, or service. The management challenge here is determining how to deal with a very large number of resources that are not controlled by a unique centralized administrator, in order to avoid a scalability weakness.

In most common architectures, computational resource monitoring is integrated with a resource selection mechanism (resource brokers and scheduling systems). It implies that for any new scheduling, a new monitoring request must also be addressed to all nodes participating in the same network domain. Therefore, regardless the monitoring mechanism implemented (pull, push, or hybrid), the resource selection mechanism will be delayed by the resource monitoring activity. In these integrated systems, we mainly found scalability issues. Moreover, the network traffic generate by these integrated systems will be increased by all the monitored data going from computational and networking resources to resource management points.

In our proposal, this phase is addressed by introducing our SNMP-based Balanced Load Monitoring Agents for Resource Scheduling (SBLOMARS) architecture. This set of autonomous monitoring agents generates real-time and statistical availability for resources and entities composing the Grid. Thanks to its conception, SBLOMARS is able to deal with any resource type, of any size, supported by any operative system.

The distributed monitoring system reaches a high level of generality by means of the integration of the Simple Network Management Protocol (SNMP) [Sta99] and thus, it has a powerful ability to handle heterogeneous operating platforms due to generalized support of such protocol. Nevertheless, we are aware that only adopting the SNMP technology is not enough to assure full systems interoperability. Data formats, objects not implemented, and objects with heterogeneous style in their data structure may create incompatibilities. Therefore, we undertook a distributed monitoring system design with an exhaustive study of the SNMP incompatibilities. Moreover, our monitoring system integrates the CISCO IOS® IP Service Level Agreements (CISCO IP SLAs), an end-to-end network-level monitoring technology allowing users to monitor end-to-end performance between switches, routers, or any IP network device.

Security is a must in a distributed monitoring system. SBLOMARS is secure because it is deployed on nodes belonging to secure virtual organizations. This creates a high level of

protection against external intruders. In addition, SBLOMARS supports the security mechanisms of SNMP version 2 (SNMPv2) to prevent any intentional or unintentional modification of monitoring parameters from local users. Nevertheless, this level of security is currently not enough to satisfy the security requirements that Grid Computing administrators demand. In the future work section we will describe our intention to improve this security mechanism.

SBLOMARS monitoring system exhibits a self-configuring design in the sense that the monitored information about resource availability is refreshed adaptively instead of on a periodic basis. Moreover, the information about the amount of resources available is presented in flexible and dynamic software structures, which can be used to monitor from simple personal computers to complex multiprocessor systems or clusters with multiple hard disk partitions. Therefore, SBLOMARS is a distributed monitoring system with minimal overhead and performance degradation of its hosting nodes. This feature ensures a high level of reliability in computing devices with minimal computational resources, such as mobile devices (i.e. laptops, PDAs, etc).

SBLOMARS also reduces the amount of traffic injected to the network. This is because it calls the SNMP daemon to collect resource availability information locally, which means that there is no traffic between SBLOMARS and the management system. On the other hand, the management system is calling the SBLOMARS monitoring agents only when needed.

SBLOMARS was successfully tested in a large-scale Grid platform, the Grid5000. Our results showed that SBLOMARS is flexible and scalable in managing large-scale heterogeneous distributed systems. Moreover, it could also be useful for small-scale distributed systems because, as we have shown in Chapter 6, regardless the number of devices to monitor, SBLOMARS remains functional and scalable.

7.2.3 Balanced Load Multi-Constraint Resource Scheduler (BLOMERS)

Resource Scheduling for Grid Computing involves determining which resources are best suited for executing a specific job, application or service. Our presented solution covers this phase by introducing the Balanced Load Multi-Constraint Resource Scheduler (BLOMERS). BLOMERS implements a heuristic approach in order to overcome the scalability problem and, by making use of the real-time and statistical resource availability information generated by SBLOMARS monitoring system, it schedules jobs in such a way that network activity and resource load remain balanced throughout the Grid.

In this scheduling system we have designed and implemented a Genetic Algorithm for improving efficiency in actual scheduling algorithms for Grid Computing. The implemented resource scheduling system based on Genetic Algorithms is able to search for a sub-optimal set

of resources for any requested Grid Service, starting the matching process from a set of resources, rather than from just one at a time. This parallelism has been introduced to avoid the Genetic Algorithm be trapped on local minima, which means that the scheduling system will be searching and matching the whole set of shared resources from the Grid Infrastructure at any given time.

It is worth mentioning that BLOMERS achieves much of its breadth by ignoring information that does not concern payoff. Other methods rely heavily on such information, and therefore solve problems where the necessary information is not available or is difficult to obtain, which causes them to break down. In this case, the information ignored must be carefully selected, and should always be modified for each type of application. We face a difficult problem due to the fact that each heuristic technique could be customized in different ways depending on their main targets and the optimization degree required for the application to execute.

The BLOMERS scheduling system deals with several conditions. Basically, it selects a set of candidate resources from a poll, keeping individual resource performance comparatively balanced in all nodes of the Grid. This condition has been added in order to satisfy computational resource load balancing. It covers multi-constraint (hard and soft constraints) service requirements, which are: user requirements (QoS, deadlines, etc.), service needs (memory, storage and software requirements), and resource-load balancing throughout the Grid.

BLOMERS improves resource load-balancing and reduces the makespan in any scheduling. This is an important contribution of our research because the most important approaches in this area only focus on reducing the makespan of the scheduling process but do not maintain resource usability in a balanced way. BLOMERS exhibits excellent performance. It is able to schedule large numbers of services in real scenarios, and guarantees a balanced load throughout the Grid. The makespan measured is less than those of the Round-Robin and Least Used algorithms.

Unlike other resource scheduling systems, BLOMERS is highly customizable. In our algorithm parameters like thresholds, end-to-end network performance, statistical resource availability information, and software available are controlled by the administrator of the scheduling system. BLOMERS allows modification of the before-mentioned parameters in order to offer better adaptability of the Grid Infrastructure being managed. BLOMERS was designed using two re-combination methods, crossover and mutation, instead of only one, like most of the current systems. The implementation of these two methods helps to deal with Virtual Organizations of different sizes, from small organizations with less than a hundred resources, to multinational organizations with thousands of resources.

On the other hand, BLOMERS is an excellent scheduling system when the number of resources is quite large. Some of our results shown in Chapter 6 indicate that with hundreds of resources to analyze, the scheduling algorithm is better than others.

7.2.4 Full System Approach Conclusions

In this thesis we have presented a solution distributed in three components. In past sections we have presented the main features, contributions, and drawbacks of each component, but we didn't explain the benefits and contributions of the approach as a whole architecture. It is important that each of the three components exhibits good behaviour in isolation. Though, when they work together they reach their maximum potential, and stronger than similar solutions [Maga07d], [Nab04] and [Maga06].

The most important benefits of the system approach conceived in this research are obtained in environments where the amount of resources is large, such as large-scale Grids [Cap05]. In summary, we claim that the main distinguishing features of the complete system are as follows:

- A fully distributed architecture consisting of three independent components: SBLOMARS, BLOMERS, and PbGRMA.
- The whole architecture is completely scalable and flexible in terms of the amount and type of elements to be managed.
- It is Web Services Resource-Framework [Ved02] oriented. This feature guarantees that our approach follows current Open Grid Forum (OGF) standards.
- It covers multi-constraint service requirements, which are: user requirements (QoS, deadlines, etc.), service needs (memory, storage, and software requirements), and resource-load balancing throughout the Grid.
- It is a distributed self-adapting system that manage a wide range of computational devices (i.e., it can handle a high level of heterogeneity).
- It is a flexible system that is able to manage from simple personal computers to robust multiprocessor systems or clusters, even when multiple hard disks and storage partitions are integrated.
- It offers a parallelism solution to multi-constraint service requests avoiding enclosure into local minima solutions.
- It improves resource load-balancing and reduces the makespan in scheduling resources. In Chapter 6 we demonstrate that BLOMERS improves effectiveness by almost 60% (this represents how close the solution selected is to the best one as established from the network administrator view) versus other scheduling algorithms when the amount of resources is greater than one hundred.

The strength of this research is that it provides evaluation results of the full process in a realistic large-scale scenario, the Grid5000 test-bed. Summarizing these results, we argue that our solution has very good performance, because it does not have a significant negative impact on the performance of the hosting nodes. We also claim that our solution is flexible and scalable. We ran several experiments with different amounts of resources, and we got similar results with all of them [Maga07d] and [Maga07e].

Besides the good properties above described we have to say that the solution presented in this thesis is complex. The system is not easy to deploy without proper knowledge of the components and the structure of the system. The interfaces between PbGRMA, SBLOMARS, and BLOMERS are based on XML standards; This means that transformation of the information from XML-based documents to policy language that commonly is named as parsing processI is complex to do and therefore it consumes high levels of time and resources.

7.3 FUTURE WORK

This section outlines possible future work on relevant topics addressed in this thesis, as well as possible alternatives to improve them.

There are many constraints included in the scheduling process, but current evaluation results do not yet include the effect of network latency and other communication impairments. As an immediate future work, we are including network performance between end-to-end edge routers as an entry parameter for our genetic algorithm. We expect that this will offer better scheduling solutions when resources with high latencies or jitter have to be managed

SBLOMARS monitoring system is ready to monitor and report end-to-end networking availability. In Chapter 4 we have presented the graphical interface implemented in this monitoring system which represents networking real-time behaviour. Unfortunately, Grid5000 is not the best scenario to deploy these experiments. It is because links between Grid nodes are private links and the amount of network traffic than these links have to support is completely insignificant. Therefore, there are no networking constraints that could modify the scheduling decisions made by BLOMERS scheduling systems. In world-wide networks, networking issues are very important. We plan to deploy new scheduling evaluations tests on different network scenarios in order to include networking constraints in the scheduling process.

As future work, we plan to improve the security aspects. Currently we are working with version two for the SNMP server configuration. We have realized that better security mechanisms should be integrated in this research. The SNMP version 3 is a string solution which keeps our architecture secure. We will implement the necessary mechanisms to integrate SNMPV3 with SBLOMARS. We are also planning to merge SBLOMARS and BLOMERS approaches with autonomic gateways [Cha05]. We expect that this conception will help distributed systems and Grids designers to evaluate and monitor more precisely the usage of their network resources.

Novel distributed paradigms are becoming increasingly popular. Cloud Computing [Ric07] is one of them. It is a computing paradigm shift in which computing is moved away from personal computers, or an individual application server, to a "cloud" of computers. Users of the cloud only need to be concerned with the computing service being requested, because the underlying details of how it is achieved are hidden. This method of distributed computing is done by pooling computer resources and managing them via software (rather than by a human). The architecture behind cloud computing is a massive network of "cloud servers" interconnected as if in a grid running in parallel, sometimes using the technique of virtualization to maximize the computing power per server. We are planning to study the reliability of our full system approach

in this new area. We think that one of the most challenging issues is the management of virtual environments and virtual resources. Very recently we have started looking at this field using Planet-lab [Bav04].

Appendix A

GRID COMPUTING TECHNOLOGY

A.1 INTRODUCTION

The Grid term was coined in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering [Fos01]. Considerable progress has since been made on the construction of such an infrastructure but the term Grid has also been conflated, at least in popular perception, to embrace everything from advanced networking to artificial intelligence. The Grid is an abstraction allowing transparent and pervasive access to distributed computing resources. Other desirable features of the Grid are that the access provided should be secure, dependable, efficient, and inexpensive, and enable a high degree of portability for computing applications.

Today Internet can be regarded as a precursor to the Grid, but the Grid is much more than just a faster version of the Internet, a key feature of the Grid is that it provides access to a rich set of computing and information services. Many of these services are feasible only if network bandwidths increase significantly. Thus, improved network hardware and protocols, together with the provision of distributed services, are both important in establishing the Grid as an essential part the infrastructure of society in this century.

Currently, the Grid is not transparent or pervasive, and computing tasks do not routinely describe their requirements, when a computing task is submitted to the Grid one or more resource brokers and schedulers decide on which physical resources the task should be executed, possibly breaking it down into subtasks that are satisfied by a number of distributed resources.

A.1.1 Virtual Organizations

The original motivation for the Grid was the need for a distributed computing infrastructure for advanced science and engineering, with a pronounced emphasis on collaborative and multi-disciplinary applications. It is now recognized that similar types of application are also found in numerous other fields, such as entertainment, commerce, finance, industrial design, and government. Consequently, the Grid has the potential for impacting many aspects of society. All these areas require the coordinated sharing of resources between dynamically changing collections of individuals and organizations.

This has led to the concept of a Virtual Organization (VO) [Fos02a], which represents an important mode of use of the Grid. The individuals, institutions, and organizations in a VO want to share the resources that they own in a controlled, secure, and flexible way, usually for a limited period of time. This sharing of resources involves direct access to computers, software, and data. Examples of VO include:

- A consortium of companies collaborating to design a new jet fighter. Among the resources shared in this case would be digital blueprints of the design (data), supercomputers for performing multi-disciplinary simulations (computers), and the computer code that performs those simulations (software).
- Physicists collaborating in an international experiment to detect and analyse gravitational waves. The shared resources include the experimental data and the resources for storing it, and the computers and software for extracting gravitational wave information from this data, and interpreting it using simulations of large-scale gravitational phenomena.

A.1.2 The Consumer Grid

Support for VOs allows computing and information resources to be shared across multiple organizations. Within a VO sophisticated authorization and access control policies may be applied at various levels (individual, group, institution, etc) to maintain the level of control and security required by the owners of the shared resources. In addition, the members of a VO are working together to achieve a common aim, although they may also have different subsidiary objectives.

The consumer grid represents another mode of use of the Grid in which resources are shared on a commercial basis, rather than on basis the basis of mutual self-interest. Thus, in the consumer grid paradigm of network-centric computing, users rent distributed resources, and although many users may use the same resources, in general, they do not have common collaborative aims. In the consumer grid, authentication and security are still important issues since it is essential to prevent a user's information, code, and data being accessible to others. But authorization to access a resource derives from the user's ability to pay for it, rather than from membership of a particular VO.

In the consumer grid, scheduling is done "automatically" by the invisible hand of economics. Supply and demand determines where jobs run through the agent negotiation process, no other form of scheduling is required. The users seek to minimize their costs subject to constraints, such as obtaining results within a certain time, and the suppliers seek to maximize their profits. For the concept of the consumer grid to become a reality the development of secure and effective computational economies is essential. In the consumer grid all resources are economic commodities. Thus, users should pay for the use of hardware for computation and storage.

If large amounts of data are to be moved from one place to another a charge may be made for the network bandwidth used. In the future it seems likely that Grid Computing will be based on a hybrid of the virtual organization and consumer grid models. In this scenario hardware, software, and data repository owners will form VOs to supply resources. Collaborating end-user organizations and individuals will also form VOs that will share resources, but also “ rent” resources outside the VO when the need arises. The consumer grid model applies to the interaction between supplier VOs and user VOs.

A.1.3 Grid Architecture

In defining Grid Architecture, we start from the perspective that effective VO operation requires that we be able to establish sharing relationships among any potential participants. Interoperability is thus the central issue to be addressed. In a networked environment, interoperability means common protocols. Hence, the Grid architecture is first and foremost protocol architecture, with protocols defining the basic mechanisms by which VO users and resources negotiate, establish, manage, and exploit sharing relationships.

A standards-based open architecture facilitates extensibility, interoperability, portability, and code sharing; standard protocols make it easy to define standard services that provide enhanced capabilities. We can also construct Application Programming Interfaces and Software Development Kits to provide the programming abstractions required to create a usable Grid. Together, this technology and architecture constitute what is often termed Middleware ("the services needed to support a common set of applications in a distributed network environment"), although we avoid that term here due to its vagueness.

A protocol definition specifies how distributed system elements interact with one another in order to achieve a specified behavior, and the structure of the information exchanged during this interaction. This focus on externals (interactions) rather than internals (software, resource characteristics) has important pragmatic benefits. VO's tend to be fluid; hence, the mechanisms used to discover resources, establish identity, determine authorization, and initiate sharing must be flexible and lightweight, so that resource-sharing arrangements can be established and changed quickly. Because VO's complement rather than replace existing institutions, sharing mechanisms cannot require substantial changes to local policies and must allow individual institutions to maintain ultimate control over their own resources.

The Grid Architecture [Fos01] and the subsequent discussion organize components into layers, as shown in Figure A.1, components within each layer share common characteristics but can build on capabilities and behaviours provided by any lower layer. In specifying the various layers of the Grid architecture, it follows the principles of the "hourglass model" [Fos03]. The narrow neck of the hourglass defines a small set of core abstractions and protocols (e.g., TCP and HTTP in the Internet), onto which many different high-level behaviours can be mapped (the top of the hourglass), and which themselves can be mapped onto many different underlying technologies (the base of the hourglass). By definition, the number of protocols defined at the neck must be small. In Grid architecture, the neck of the hourglass consists of Resource and Connectivity protocols, which facilitate the sharing of individual resources.

Protocols at these layers are designed so that they can be implemented on top of a diverse range of resource types, defined at the Fabric layer, and can in turn be used to construct a wide range of global services and application-specific behaviours at the Collective layer so called because they involve the coordinated ("collective") use of multiple resources.

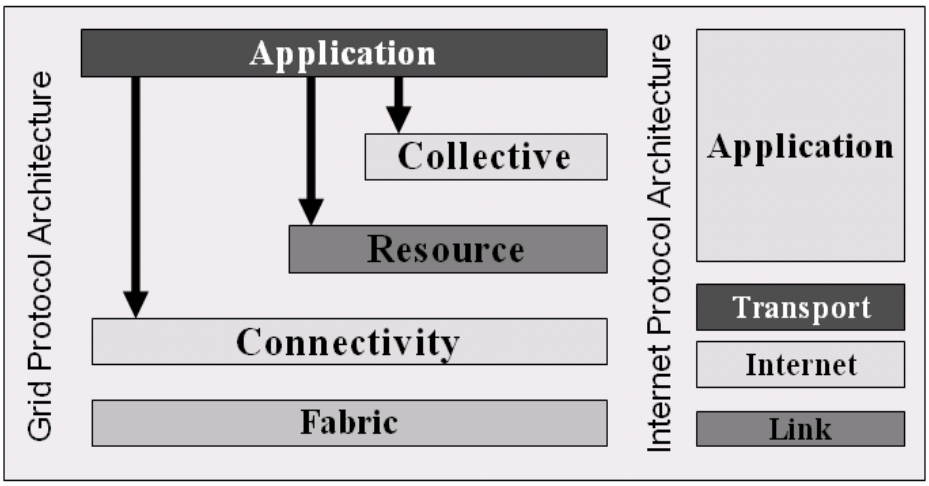

Figure A.1 The GRID Architecture

A.2 GRID RESOURCE MANAGEMENT TECHNOLOGIES BACKGROUND

Over the past several years there have been a number of projects aimed at building "production" Grids. These Grids are intended to provide identified user communities with a rich, stable, and standard distributed science environment. By "standard" and "Grids" we specifically mean Grids based on the common practice and standards coming out of the Global Grid Forum (GGF) (www.gridforum.org) [GGF07].

There are a number of projects around the world that are in various stages of putting together production Grids that are intended to provide this sort of persistent cyber infrastructure for science. These are the UK's e-Science program [Uke07], the European DataGrid [Dat07], several Grids under the umbrella of the DOE Science Grid [Doe07], and (at a somewhat earlier sage of development) the Asia-Pacific Grid [Asp07].

In addition to these basic Grid infrastructure projects, there are a number of well advanced projects aimed at providing the sorts of higher-level Grid services that will be used directly by the scientific community. Some of the important infrastructure components of these Grids are described below.

A.2.1 Grid Middleware Projects

The following projects have successfully provided U.S. and international projects with the advanced tools to easily access numerous grid functionalities, such as computation, visualization, and storage resources. You can interact with various grids or have one customized to work with your own grid.

We are presenting these projects because they are performing one or more of the management activities in the GRM process. Unfortunately, most of these approaches work under the assumption that user is the one in charge of individually select the resource to use in order to execute their applications.

Globus Toolkit: Globus Toolkit [Bor05] is a collection of tools that provides the basic services and capabilities like security, resource management, information services, etc. required for grid computing. Resource Management System of Globus consists of resource brokers, resource co-allocators and resource manager or GRAM [Cza98]. The resource requests are specified in extensible resource specification language (RSL). Globus has a decentralized scheduling model. Scheduling is done by application level schedulers and resource brokers. Application specific brokers translate the application requirements into more specific resource specification. Resource Brokers are responsible for taking high-level RSL specification and transforming them into more concrete specification (this process is called specialization).Requests can be passed to multiple Brokers. Transformations done by the brokers' results in a request in which the locations of the resources are completely specified.

The Resource Brokers discover resources by querying the information service (MDS) for resource availability. MDS is a LDAP based network directory (Metacomputing Directory Services). MDS consists of two components Grid Index Information service (GIIS) and Grid resource information service (GRIS). GRIS provides resource discovery services. GIIS provides a global view of the resources by pulling information from the GIIS's. Resource information on the GIIS's is updated by push dissemination. Globus has a hierarchical name space organization.

The transformed resource requests from resource brokers are passed to the co-allocator. Co-allocator takes care of multi-requests, multi request is a request involving resources at multiple sites which need to be used simultaneously, and passes each component of the request to appropriate resource manager and then provides a means for manipulating each resultant set of managers as a whole. The Co-allocation of resources is done by the DUROC component Globus. The resource manager interacts with local resource management systems to actually schedule and execute the jobs. The implementation of the resource manager in

Globus is called GRAM. GRAM authenticates the resource requests and schedules them on the local resource manager. Each user is associated with a UHE (user hosting environment) on the execution machine. All the jobs from a user are directed to the user's UHE, which starts up a new Managed Job Factory service (MJFS) instance for every job. The MJFS communicated with the clients by starting up two instances of File Stream Factory Service (FSFS) for standard input and output. MJFS and FSFS are persistent services. When a system starts up after a fault all the UHE which were running before crash are started up. There is also a Sweeper task which runs every two hours and recovers any crashed UHE. The information about the UHE is obtained from gridMapfile which stores the status (Active, Inactive) of the UHE's in the system, if a nonexistent UHE is marked active in the gridMapfile it indicated that the UHE crashed. After a UHE is restarted all the persistent services in it (MJFS and FSFS) are recovered.

GLite: It is the next generation middleware for grid computing, born from the collaborative efforts of more than 80 people in 12 academic and industrial research centers as part of the EGEE Project. gLite [Gli07] provides a bleeding-edge best-of-breed framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet.

Ninf-G: It is a Japanese project developing programming middleware which enables users to access various resources, such as hardware, software, and scientific data on the grid with an easy-to-use interface. Ninf-G [Nak99] is open source software that supports development and execution of grid-enabled applications using Grid Remote Procedure Call (GridRPC) on distributed computing resources.

NorduGrid: NorduGrid Middleware [Nor07] is also known as Advanced Resource Connector (ARC), is an open source software solution distributed under the GPL license, enabling production-quality computational and data grids. ARC provides a reliable implementation of the fundamental grid services, such as information services, resource discovery and monitoring, job submission and management, brokering and data management, and resource management. Most of these services are provided through the security layer of the GSI. The middleware builds upon standard open source solutions like OpenLDAP, OpenSSL, SASL and Globus Toolkit (GT) libraries.

OGSA-DAI: The OGSA-DAI [Fos02a] project focuses on the development of middleware to assist with the access and integration of data from separate sources through the grid. The

project works closely with the Globus, OMII-Europe, NextGRID, SIMDAT, and BEinGRID, ensuring that the OGSA-DAI software works in a variety of grid environments.

A.2.2 Grid Monitoring Projects

There are lot of works has been done in Grid Monitoring. For time limitation, we can not discuss all of them, but definitely the best summary so far, is presented in [Zan05]. Currently, SNMP-based monitoring agents have been implemented by many researchers to solve different problems [Yan02] [Maga04], each system has its own strengths and weaknesses. However, for resource monitoring and management, we believe that none fulfil the criteria that are desired in emerging generation networks [NGG07]. GridLab [GridLab] aims to enable applications to fully exploit dynamically changing computational resources. In order to accomplish this, a variety of mechanisms have been developed such as notifications about changes of job states, complex workflow support, multi-criteria user-preference driven and prediction-based selection of the best resources, submission of jobs with time constraints, and many others. The capability of a dynamic adaptation has been achieved using job migration and pointing techniques.

Monitoring resources and applications is the key to the success of grids. Through an easy-to-use interface, these sophisticated tools help users gather, catalogue, and monitor various types of resources. Moreover, systems administrators are also able to monitor the health of their grids. These evolving grid projects list a few of the open source options.

Open Grid Forum – Grid Monitoring Working Group: The Global Grid Forum (GGF) [GFF07] has proposed a general architecture for monitoring Grids. They have defined important concepts, which have been used along this chapter. The first of them is a Producer. Basically, we have used this concept to define our monitoring agents. A monitoring agent is a program that generates a time-stamped performance-monitoring event. The second one is a directory service, which is used to publish the location of the information provider and its associated sensors. This allows the users to discover information provider and it is known as Directory Service. Finally, the Information Consumer, which is any program that receives data from a producer. We believe important to mention this approach because SBLOMARS follows the main recommendations from this forum.

Grid Resource Manager (GridRM): One of the most similar approaches to SBLOMARS has been presented in [Bak05]. GridRM is also a generic monitoring architecture that has been specifically designed for the Grid. It was developed joining several technologies and standards like Java (applets, servlets and JDBC), SQL Databases, Grid Monitoring Architecture and it

follows several recommendation by Open Grid Forum. SBLOMARS could be more competitive base on its low resources consumption and its availability to offer at any moment reliable information regarding availability of computational resources.

GridRM uses the Mercury Grid Monitoring System to deliver trace data to the host of visualization and PROVE [Bal01] visualizes trace information on-line during the execution of the grid applications. Unfortunately, they are oriented to Application Monitoring instead of Resource Monitoring and their two fixed layers structure is a scalability lack. GRM is a semi-on-line monitor that collects information about an application running in a distributed heterogeneous system and delivers the collected information to the PROVE visualisation tool. The information can be either event trace data or statistical information of the application behaviour. Semi-on-line monitoring means, that any time during execution all available trace data can be required by the user and the monitor is able to gather them in a reasonable amount of time. GirdRM consists of three main components: client library, local monitor process and main monitor process.

PROVE: It has been developed for performance visualisation of Tape/PVM trace files. It supports the presentation of detailed event traces as well as statistical information of applications [Bal01]. It can work both off-line and semi-on-line and it can be used for observation of long-running distributed applications. Users can watch the progress of their application and realise performance problems in it. PROVE communicates with the main monitor of GRM and asks for trace collection periodically. It can work remotely from the main monitor process. With the ability of reading new volumes of data and removing any portion of data from its memory, PROVE can observe application for arbitrary long time.

The Ganglia Distributed Monitoring System: The Ganglia distributed monitoring system [Mas04] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

Each node monitors its local resources and sends multicast packets containing monitoring data on a well-known multicast address whenever significant updates occur. Applications may also send on the same multicast address in order to monitor their own application-specific metrics. Ganglia distinguishes between built-in metrics and application-specific metrics through a field in the multicast monitoring packets being sent. All nodes listen for both types of metrics on the well-known multicast address and collect and maintain monitoring data for all other nodes. Thus, all nodes always have an approximate view of the entire cluster's state and this state is easily reconstructed after a crash.

Monitoring Agents using a Large Integrated Services Architecture: Monitoring Agents using a Large Integrated Services Architecture (MonAlisa) [Leg04] is a distributed monitoring service based on JINI/JAVA and WSDL/SOAP technologies, which provides monitoring information from large and distributed systems to higher level services that require such information. It could be argued that JINI is using multicast, which is not always available, and places scalability limits. Although, MonAlisa is a well justified flexible system, it runs remote scripts in Grid resources get behavioural information, this mainly causes an extra traffic in the network. Unlike this approach, SBLOMARS do not add monitoring traffic between resources forming the Grid. SBLOMARS offers end-to-end network monitoring with less impact in the network performance than MonAlisa does.

Globus Heartbeat Monitor and Monitoring and Discovery Service: The Globus Heartbeat Monitor (HBM) [Bor05] was designed to provide a simple but reliable mechanism for detecting and reporting the failure (and state changes) of Globus system processes and application processes. The HBM module has been deprecated in the Globus toolkit. A daemon ran on each host gathering local process status information. A client was required to register each process that needed monitoring. Periodically, the daemon would review the status of all registered client processes, update its local state and transmit a report (on a per process basis) to a number of specific external data collection daemons. These data collecting daemons provided local repositories that permitted knowledge of the availability of monitored components based on the received status reports. The daemons also recognised status changes and notified applications that registered an interest in a particular process. The HBM was capable of process status monitoring and fault detection. HBM was unable to monitor resource performance, just availability, was based on a non-standard message format and required component daemons to be ported to all the available platforms used with a Grid.

The Monitoring and Discovery Service (MDS), constitutes the information infrastructure of the Globus Toolkit [Bor05]. Globus was designed and implemented as part of the Open Grid Services Architecture (OGSA) [Fos02a]. The Lightweight Directory Access Protocol (LDAP) [Ldap07] is adopted as a data model and representation, a query language and a transport protocol for MDS. However, LDAP features a non-declarative query interface that requires knowledge of the employed schema. In addition, the performance of OpenLDAP's update operation, which is by far the most frequently used, has been very much criticized.

NetLogger: Distributed System Performance Tuning and Debugging: NetLogger [Tie02] is both a methodology for analyzing distributed systems, and a set of tools to help implement the methodology. It provides tools for distributed application performance monitoring and analysis. It provides tools for distributed application performance monitoring and analysis. The toolkit enables users to debug, tune and detect bottlenecks in distributed applications. NetLogger components include client libraries, data monitoring, storage, retrieval and visualisation tools, and an open messaging format. The NetLogger messages can comprise of string, binary or XML encoded formats. The client library allows developers to add calls to existing source code so that monitoring events can be generated from their applications (events can be sent to file, network server, syslogd, or memory). The visualisation tool provides a means for event logs to be analysed. The storage and retrieval tools include a daemon to combine NetLogger events from multiple sources to a single central host and an event archive system. NetLogger provides a means to debug, tune and detect bottlenecks in distributed applications with a common messaging format. NetLogger requires source code modification, there is a single data collection repository, so scalability will be an issue in a Grid environment, and it appears that there is no security within the system.

Resource Monitoring System (REMOS): The Remos (REsource MOnitoring System) [Dew04] aims to allow network-aware applications to obtain relevant information about their execution environment. It defines a uniform heterogeneous interface that addresses statistical information and efficiency for these environments. Remos aims to balance information accuracy (best-effort or statistical information) with efficiency (providing a query-based interface), to manage monitoring overheads. Remos consists of multiple (SNMP-based) collectors, which gather information about the network. Remos permits applications to obtain monitoring information in a portable manner, using platform independent interfaces, however it appears to lack any security mechanisms.

Java Agents for Monitoring and Management (JAMM): Finally, Java Agents for Monitoring and Management [Law00] is a distributed set of sensors that collect and publish monitoring information regarding computational systems. These systems have been presented for different frameworks and some of them are not available any more. Gateways allow clients to control or subscribe to a number of monitoring sensors, provide multiplexing/de-multiplexing and information filtering tasks. Loggers control the operation of sensors, and manage subscriptions for sensor output. Sensors execute on host systems, parse output from processes and transmit monitoring information to subscribers. A data collector combines data from multiple sensors into a single file for real-time visualization and analysis. A centralised directory service is required for event consumers to locate (registered) event producers. JAMM uses a subscription-based, event notification approach for monitoring processes executing on hosts. The central directory service is implemented using a hierarchy of replicated LDAP servers. Java based sensors can be implemented as active objects, permitting sensors to be added, removed dynamically or reconfigured, while non-Java sensors must be installed on each host. With JAMM, it appears that sensors are key aspects of concern. If the sensor is pure Java then there is only a limited amount of real system data that can be collected without resorting to native system calls.

A.2.3 Grid Resource Schedulers

A number of schedulers for grid computing systems have been developed [Fan06]. However only a few (Nimrod-G, GrADS and Condor would seem to (a) be at a relatively advanced stage of development and (b) have wide spread operational deployment by current Grid operators. It is interesting to note that, at the time of writing the Globus [Bor05] project is the current favourite grid computing toolkit, having been adopted by IBM, HP, etc. and thus many of the smaller scheduling projects have now either been integrated into Globus, or abandoned in favour of it.

Globus Resource Management Architecture (GRMA): Globus resource management architecture consists of information service that is responsible for providing information about the current availability and capability of resources, co-allocator which is responsible for coordinating the allocation and management of resources at multiple sites, manager that is responsible for taking RSL (Resource Specific Language) specification, and GRAM [Glo07a] (grid resource allocation management) that is responsible for managing local Iesource. The previous resource management approaches as Globus don’t solve an optimal resource selection problem that is caused when the number of resources that satisfy user’s demand is

much more than the number of resources that User needs. Also, they don't provide fault tolerance service to guarantee the efficient job execution after job allocation. In Globus, a noticeable flaw is the lack of support for fault tolerance. To date, grid applications have either ignored failure issues or have implemented fault detection and response behavior completely within the application. The support for fault tolerance has consisted mainly of fault detection services or monitoring system.

Globus Architecture for Reservation and Allocation (GARA): The most well known technology available to reserve and allocate low-level resources such as memory, processing or storage is the Globus Architecture for Reservation and Allocation (GARA) [Fos99]. It enlarges the Globus resource management architecture basically in two ways: the first one is to generate an architecture with the ability of co-reserving resources and the second one is to generate a generic resource object in order to manage heterogeneous resources.

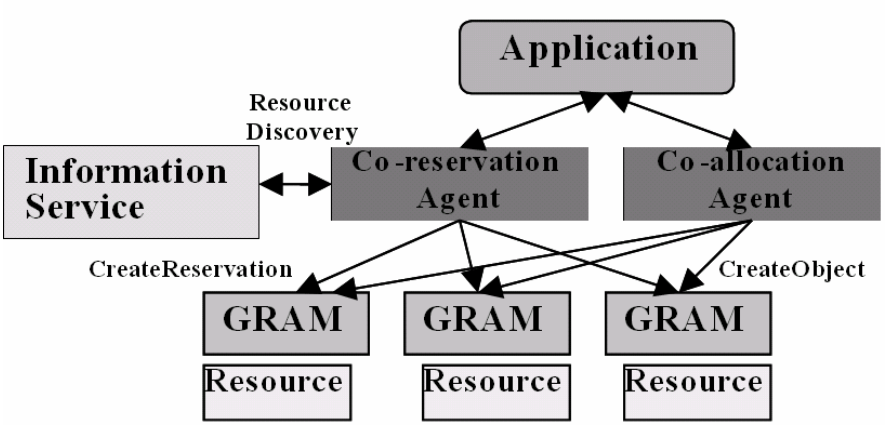

Figure A.2 The GARA Architecture

The GARA architecture allows applications appropriate access to end-to-end Quality of Service (QoS). To do so, it provides a mechanism for making QoS reservation for different types of resources, including disks, computers and networks. GARA provides management for separately administered resources. In Figure A.2 we show the GARA's architecture, which consists of three main components, the information service, local resource managers and co-allocation/reservation agents. The information service allows applications to discover resource properties such as current and future availability. Local resource managers have been implemented for a variety of resource types, this explains the use of term "resource manager" rather than the more specific "bandwidth broker", although each one implements reservation, control and monitoring operations for a specific resource.

The reservation and allocation agents compute the resource requirements and send the request to the Globus Resource Allocation Manager. The GRAM takes the request, authenticates it and if successful forwards it to the local scheduler in order to allocate or serve the resource and finally returns the job handle to GARA.

The support for heterogeneous resources is provided by a generic resource object that encapsulates network flows, memory blocks, disk blocks, and even processes. The support for advanced reservation is allowed by separating the reservation from allocation. For immediate resource allocation, the requested allocation is performed at the time of reservation, but in the case of advance reservation only a reservation handle is returned and the resources need to be reserved at the service start time. A new entity that is called Co-reservation Agent provides this advanced functionality. Its function is similar to the co-allocation agent one, except that after calculating the resource requirement for advanced reservation, it does not allocate but simply reserves the resources. GARA provides a simple, uniform interface for making advance reservations for any type of supported resource. The API provides functions for creating, modifying, binding, claiming, canceling, and monitoring reservations. Although it is as uniform as possible, different resources require different parameters. To accommodate these different needs within a single API, the create function call accepts a Resource Specification Language (RSL) string to specify the parameters for single reservation.

Condor-G: The Condor-G system [Tha05] combines the inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource management methods of Condor to allow the user to harness multi-domain resources as if they all belong to one personal domain. Condor-G handles all aspects of discovering and acquiring appropriate resources regardless of their location; initiating, monitoring and managing execution on those resources; detecting and responding to failure and notifying the user of termination.

Condor-G is a resource management system designed to support high-throughput computations by discovering idle resources on a network and allocating those resources to application tasks. The main function of Condor-G is to allow utilization of machines that otherwise would be idle thus solving the wait-while-idle problem. A cluster of workstations managed by Condor-G is called a condor pool. Jobs submitted by the users are queued by Condor-G and scheduled on available machines in the pool transparently to the user.

Condor-G selects a machine from the pool to run a user job, it can also migrate a running job from one machine to another until it is completed. Condor-G has a centralized scheduling model. A machine is the condor system (Central Manager) is dedicated to scheduling. Each condor work station submits the jobs in its local queue to the central scheduler which is

responsible for finding suitable resources for the job execution. The information about suitable available resources to run the job (execution machine information) is returned to the job submission machine. A shadow process is forked on the submission machine for each job, which is responsible for contacting and staging the job on the execution machine and monitoring its progress. Condor-G supports pre-emption of running jobs, if the execution machine decides to withdraw the resources Condor-G can pre-empt the job and schedule it on another machine thus providing for resource owner autonomy. The resource information needed for making scheduling decisions is also stored on the central Manager.

Nimrod-G: The Nimrod-G [Buy00] is the most interesting of all the current grid meta-schedulers. As part of the GRid Architecture for Computational Economy (GRACE) project. Nimrod-G also supports quality of service based scheduling (e.g. on the basis of deadlines and budgets). This grid-enabled resource management and scheduling mechanism provides a simple declarative parametric modelling language for expressing parametric experiments. Specifically, it supports user-defined deadline and budget constraints for schedule optimizations and manages the supply and demand for resources in the Grid using a set of resource trading services.

Nimrod-G is grid-enabled resource management and scheduling system based on the concept of computational economy. It was designed to run parametric applications on computational grid. It uses the middleware services provided by Globus Toolkit but can also be extended to other middleware services. Nimrod-G uses the MDS services for resource discovery and GRAM APIs to dispatch jobs over grid resources. The users can specify deadline by which the results of there experiments are needed. Nimrod-G broker tries to find the cheapest resources available that can do the job and meet the deadline. Nimrod uses both static cost model (stored in a file in the information database) and dynamic cost model (negotiates cost with the resource owner) for resource access cost trade-off with the deadline.

Legion: Legion [Cha99] is a reflective, object-based operating system for the Grid. It offers the infrastructure for grid computing. Legion provides a framework for scheduling which can accommodate different placement strategies for different classes of applications. Scheduler in Legion has a hierarchical structure. Users or active objects in the system invoke scheduling to run jobs, higher level scheduler schedules the job on cluster or resource group while the local resource manger for that domain schedules the job on local resources. Scheduling in Legion is placing objects on the processors. The resource namespace is graph based.

Scheduling framework acts as mediator to find a match between the placement requests and processors. When a job request is submitted, appropriate scheduler for the job is selected from the framework. Selection of the scheduler can be made by the user explicitly or attributes of the Class Object of the application can be used to specify the scheduler for the objects of that class. Object specific placement constraints are also specifies as attributes on the class object. Resource owners also specify security and resource usage policies using class Object attributes. Scheduler Object uses this information in making scheduling decision. Co-allocation of resources is not supported. The enactor object is responsible for enforcing the schedule generated by the scheduler object; more than one schedule is generated, if a schedule fails another one is tried until all the schedules are exhausted. Information about resources in the grid is stored in database object called a collection. For Scalability there could be more than collection object and collections can send and receive data from each other. Information is obtained from resources either by pull or push mechanism. Users or Schedulers query the collection to obtain resource information.

Grid Resource Agreement and Allocation Protocol Working Group: The International Global Grid Forum (GGF) [Gra07a] is working on better solutions for scheduling applications with QoS through its Grid Resource Agreement and Allocation Protocol Working Group (GRAAP-WG), which is concerned with various issues relating to resource scheduling and resource management in Grid environments. To make use of distributed resources within the Grid at the same time to solve a problem a Super-Scheduling Service is necessary. Through this service access to and use of various resources managed by different schedulers in use within a Grid will be possible. The Grid Resource Allocation Agreement Protocol Working Group addresses the protocol between a Super-Scheduler (Grid Level Scheduler) and local Schedulers necessary to reserve and allocate resources in the Grid as a building block for this service. This working group has produced a "state of the art" document, laying down properties for advanced reservation in Grids [Gra07b]. This is a document which is designed to catalogue the advance reservation functionality which is available in current scheduling systems.

Appendix B

GENETIC ALGORITHMS BACKGROUND

B.1 GENETIC ALGORITHMS BACKGROUND

Genetic Algorithms (GAs) were firstly proposed by John Holland in 1975 [Hol75]. Genetic Algorithms are search algorithms based on the mechanics of the natural selection process (biological evolution). The most basic concept is that the strong tend to adapt and survive while the weak tend to die out. That is, optimization is based on evolution, and the "Survival of the fittest" concept.

Every organism has a set of rules, describing how that organism is built up from the tiny building blocks of life. These rules are encoded in the genes of an organism, which in turn are connected together into long strings called chromosomes. Each gene represents a specific trait of the organism, like eye colour or hair colour, and has several different settings. For example, the settings for a hair colour gene may be blonde, black or auburn. These genes and their settings are usually referred to as an organism's genotype. The physical expression of the genotype - the organism itself - is called the phenotype.

When two organisms mate they share their genes. The resultant offspring may end up having half the genes from one parent and half from the other. This process is called recombination. Very occasionally a gene may be mutated. Normally this mutated gene will not affect the development of the phenotype but very occasionally it will be expressed in the organism as a completely new trait.

GAs have the ability to create an initial population of feasible solutions [Rev95], and then recombine them in a way to guide their search to only the most promising areas of the state space. Each feasible solution is encoded as a chromosome (string) also called a genotype, and each chromosome is given a measure of fitness via a fitness (evaluation or objective) function. The fitness of a chromosome determines its ability to survive and produce offspring. A finite population of chromosomes is maintained.

A genetic algorithm is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. They are a particular class of evolutionary algorithms (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated

due to a maximum number of generations, a satisfactory solution may or may not have been reached.

The most important parameters in GAs:

- Entities Encoding
- Population Size
- Evaluation Function
- Selection Methods
- Evolution Methods

B.1.1 Entities Encoding

A GA manipulates populations of chromosomes, which are string representations of solutions to a particular problem. A chromosome is an abstraction of a biological DNA chromosome. A particular position or locus in a chromosome is referred to as a gene and the letter occurring at that point in the chromosome is referred to as the allele value or simply allele. Any particular representation used for a given problem is referred to as the GA encoding of the problem. The classical GA uses a bit-string representation to encode solutions. Bit-string chromosomes consist of a string of genes whose allele values are characters from the alphabet {0,1}. For problems where GAs are typically applied, solution sets are finite but so large that brute-force evaluation of all possible solutions is not computationally feasible. It is not uncommon for a GA to be operating on bit strings of length 100, giving a solution space consisting of 2100–1030 individuals.

The interpretation of these strings is entirely problem dependent. For example, a bit string of length 20 might be used to represent a single integer value (in standard binary notation) in one problem, whereas, in another, the bits might represent the presence or absence of 20 different factors in a complex process. It is a strength of GAs that common representations can be used in this way for a multiplicity of problems, allowing the development of common processing routines and operators and making it faster and easier to apply GAs to new situations. On the other hand, the consequence is that the chromosome encoding alone will contain only limited problem-specific information. Much of the meaning of a specific chromosome for a particular application is encoded in the second component of a GA, the fitness function.

B.1.2 Population Size

An initial population is created from a random selection of solutions (which are analagous to chromosomes). This is unlike the situation for Symbolic AI systems, where the initial state in a problem is already given instead. Determining the size of the population is a crucial factor. Choosing a population size too small increases the risk of converging prematurely to a local minima, since the population does not have enough genetic material to sufficiently cover the problem space. A larger population has a greater chance of finding the global optimum at the expense of more CPU time. The population size remains constant from generation to generation.

B.1.3 Evaluation Function

A value for fitness is assigned to each solution (chromosome) depending on how close it actually is to solving the problem (thus arriving to the answer of the desired problem). (These "solutions" are not to be confused with "answers" to the problem, think of them as possible characteristics that the system would employ in order to reach the answer.).

The fitness function is a computation that evaluates the quality of the chromosome as a solution to a particular problem. By analogy with biology, the chromosome is referred to as the genotype, whereas the solution it represents is known as the phenotype. The translation process can be quite complicated. In timetabling and manufacturing scheduling GAs, for example, a chromosome is translated into a timetable or set of scheduled activities involving large numbers of interacting resources. The fitness computation will then go on to measure the success of this schedule in terms of various criteria and objectives such as completion time, resource utilisation, cost minimisation and so on. This complexity is reminiscent of biological evolution, where the chromosomes in a DNA molecule are a set of instructions for constructing the phenotypical organism. A complex series of chemical processes transforms a small collection of embryonic cells containing the DNA into a full-grown organism, which is then “evaluated” in terms of its success in responding to a range of environmental factors and influences.

B.1.4 Selection Methods

A GA uses fitness as a discriminator of the quality of solutions represented by the chromosomes in a GA population. The selection component of a GA is designed to use fitness to guide the evolution of chromosomes by selective pressure. Chromosomes are therefore

selected for recombination on the basis of fitness. Those with higher fitness should have a greater chance of selection than those with lower fitness, thus creating a selective pressure towards more highly fit solutions. Selection is usually with replacement, meaning that highly fit chromosomes have a chance of being selected more than once or even recombined with themselves. The traditional selection method used is Roulette Wheel (or fitness proportional) selection. This allocates each chromosome a probability of being selected proportional to its relative fitness, which is its fitness as a proportion of the sum of fitnesses of all chromosomes in the population. There are many different selection schemes. Random Stochastic Selection explicitly selects each chromosome a number of times equal to its expectation of being selected under the fitness proportional method. Tournament Selection first selects two chromosomes with uniform probability and then chooses the one with the highest fitness. Truncation Selection simply selects at random from the population having first eliminated a fixed number of the least fit chromosomes.

Those chromosomes with a higher fitness value are more likely to reproduce offspring (which can mutate after reproduction). The offspring is a product of the father and mother, whose composition consists of a combination of genes from them (this process is known as "crossing over".

B.1.5 Evolution Methods

GAs use probabilistic rules to evolve a population from one generation to the next. The generations of the new solutions are developed by genetic recombination operators:

- **Biased Reproduction:** Selecting the fittest to reproduce. This method is the hardest of all of them. In this case just the best organisms are selected but this methodology could affect further generations or the new offspring. It is really not very recommended because there is a great level of probability to fall down into a local minima.

- **Crossover:** Combining parent chromosomes to produce children chromosomes. It combines the "fittest" chromosomes and passes superior genes to the next generation. Mate each string randomly using some crossover technique. For each mating, randomly select the crossover position(s). (Note one mating of two strings produces two strings. Thus the population size is preserved).

- **Mutation:** Altering some genes in a chromosome. It ensures the entire state-space will be searched, (given enough time) and can lead the population out of a local minima. Mutation is performed randomly on a gene of a chromosome. Mutation is rare, but extremely important. As an example, perform a mutation on a gene with probability .005. If the population has g total genes (g = string length * population size) the probability of a mutation on any one gene is 0.005g, for example. This step is a no-op most of the time. Mutation insures that every region of the problem space can be reached. When a gene is mutated it is randomly selected and randomly replaced with another symbol from the alphabet.

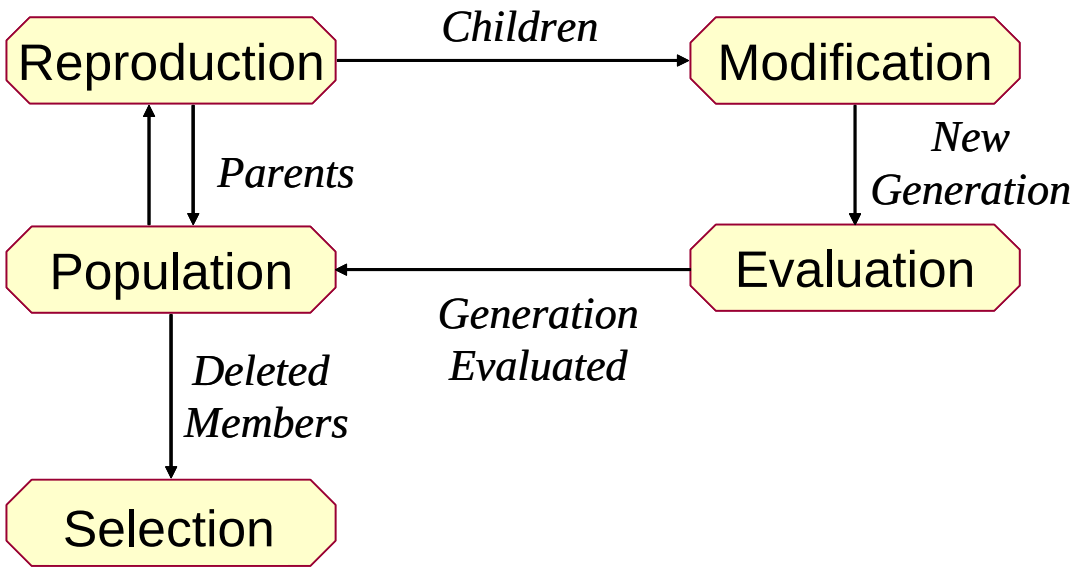

Figure B.1 The Cycle of Life of the Genetic Algorithms

After recombination, resultant chromosomes are passed into the successor population. The processes of selection and recombination are then iterated until a complete successor population is produced. At that point the successor population becomes a new source population (the next generation). The GA is iterated through a number of generations until appropriate topping criteria are reached. These can include a fixed number of generations having elapsed, observed convergence to a best-fitness solution, or the generation of a solution that fully satisfies a set of constraints.

There are several evolutionary schemes that can be used [Bac93], depending on the extent to which chromosomes from the source population are allowed to pass unchanged into the successor population. These range from complete replacement, where all members of the successor population are generated through selection and recombination to steady state, where the successor population is created by generating one new chromosome at each generation and using it to replace a less-fit member of the source population.

The choice of evolutionary scheme is an important aspect of GA design and will depend on the nature of the solution space being searched. A widely used scheme is replacement-with-elitism. This is almost complete replacement except that the best one or two individuals from the source population are preserved in the successor population. This scheme prevents solutions of the highest relative fitness from being lost from the next generation through the nondeterministic selection process. If the new generation contains a solution that produces an output that is close enough or equal to the desired answer then the problem has been solved. If this is not the case, then the new generation will go through the same process as their parents did. This will continue until a solution is reached (Figure B.1).

B.2 THE STANDARD GENETIC ALGORITHM

The standard genetic algorithm proceeds as follows:

1. Choice of Alphabet.
2. Choose initial population
3. Evaluate the fitness of each individual in the population
4. Repeat
    1. Select best-ranking individuals to reproduce
    2. Breed new generation through crossover and mutation (genetic operations) and give birth to offspring
    3. Evaluate the individual fitnesses of the offspring
    4. Replace worst ranked part of population with offspring
5. Until <terminating condition>

An example of the distribution of individuals (solution) from one generation to the next one is shown in Figure B.2 and Figure B.3.

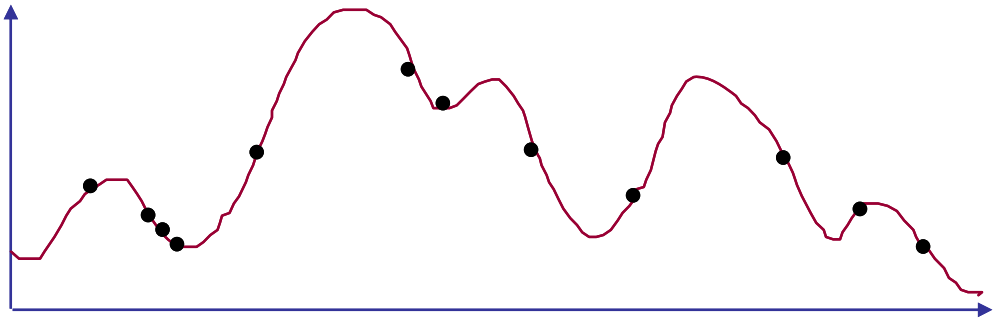

Figure B.2 Distribution of Individuals in Generation 0

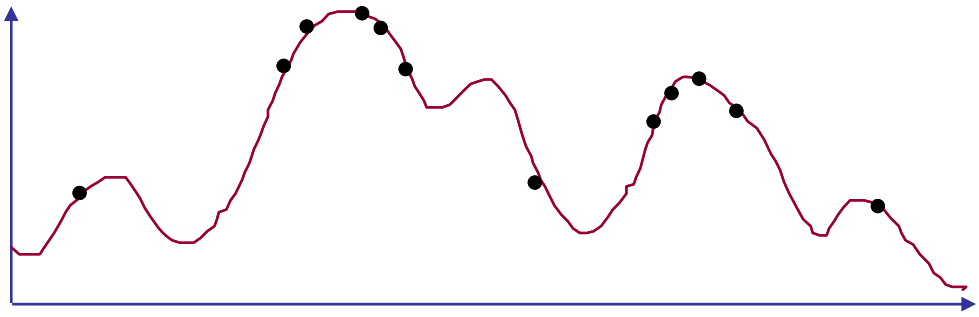

Figure B.3 Distribution of Individuals in Generation N

Appendix C

PUBLICATIONS OF THE AUTHOR

- **Edgar Magana**, Joan Serrat, Laurent Lefèvre and Masum Hasan. "Service Oriented Architecture for Large-Scale Grids", Computer Communications Journal, Editorial Elsevier 2008 (submitted and pending of acceptance).

- **Edgar Magana**, Laurent Lefèvre, Masum Hasan, and Joan Serrat. "SNMP-based Monitoring Agents and Heuristic Scheduling for large scale Grids", Grid computing, high-performAnce and Distributed Applications (GADA'07), Vilamoura, Algarve, Portugal, November 2007

- **Edgar Magana**, Laurent Lefèvre and Joan Serrat, “SBLOMARS: SNMP-based Balanced Load Monitoring Agents for Resource Scheduling in Grids”, GCA'07: The 2007 International Conference on Grid Computing and Applications. Las Vegas, Nevada, USA. June 25-28, 2007.

- **Edgar Magana**, and Joan Serrat, “Distributed and Heuristic Policy-based Resource Management System for Large-scale Grids”, AIMS 2007: The ACM Conference in Autonomous Infrastructure, Management and Security. Oslo University College, Norway. June 21-22 2007.

- **Edgar Magana**, Masum Hasan and Joan Serrat, "BLOMERS: Balanced Load Multi-Constrained Resource Scheduler”, Third International Conference on Networking and Services (ICNS 2007). Athens, Greece. June 19-25, 2007 -

- **Edgar Magana**, Laurent Lefevre and Joan Serrat, "Autonomic Management Architecture for Flexible Grid Services Deployment Based on Policies", ARCS'07: Architecture of Computing Systems, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland March 12-15, 2007.

- **Edgar Magana** and Joan Serrat, "Services and Applications Management Middleware for Autonomic Grid Computing", 1st IEEE/IFIP International Workshop on Autonomic Grid Networking and Management (Manweek2005), Universitat Politècnica de Catalunya, Barcelona, Spain. October 24-26, 2005.

- **Edgar Magana** and Joan Serrat, "QoS Aware Policy-Based Management Architecture for Service Grids", 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE/ETNGRID 2005). Linkoping University, Sweden, June 13-15, 2005.

- **Edgar Magana** and Joan Serrat, "QoS Aware Resource Management Architecture for OGSA Services Deployment", 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005). Nice, Acropolis, France. June 19-25, 2007. (Poster)

- **Edgar Magana** and Joan Serrat, "Adaptive Management Middleware for Grid Services and Applications Based on Policies", Workshop on Adaptive Grid Middleware (AGridM2004), Antibes Juan-les-Pins, France, September 29, 2004.

- **Edgar Magana**, Epifanio Salamanca and Joan Serrat, "A Proposal of Policy-Based System Architecture for Grid Services Management", Active and Programmable Grids Architectures and Components (APGAC'04). Kraków, Poland. June 19-25, 2004.

- Y. Nikolakis, **Edgar Magana**, M. Solarski, A. Tan, E. Salamanca, Joan Serrat, C. Brou, A. Galis, "A Policy-Based Management Architecture for Flexible Service Deployment in Active Networks", International Working Conference on Active Networks (IWAN03) Japan 2003.

- C. Tsarouchis, S. Denazis, Chiho Kitahara, J. Vivero, E. Salamanca, **Edgar Magana**, A. Galis, J. Mañas, Y. Carlinet, et.al., "Policy-Based Management Architecture for Active and Programmable Networks", IEEE Network, May 2003, Vol.17, No.3.

References

A

[Ahm01] I. Ahmad, Y.K. Kwok and M. Dhodhi. “Scheduling parallel programs using genetic algorithms”. John Wiley and Sons, New York, USA, 2001.

[Ali04] R. J. Al-Ali, K. Amin, G. Laszewski, O. Rana, et al. “Analysis and Provision for Distributed Grid Applications”. Journal of Grid Computing, Kluwer, 2004.

[All02] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. “Data management and transfer in high-performance computational Grid environments”. Parallel Computing Journal, 28(5):749–771, 2002.

[And00] I. Anderson, "A First Couse in Combinatorial Mathematics". Clarendon Press, Oxford.

[ANS] The ASN.1 Consortium. Web Page: http://www.asn1.org/consortium.htm

[Asp07] Asia-Pacific Grid http://www.apgrid.org/

B

[Bac93] T. Bäck and H. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization", Evolutionary Computation, Vol. I Issue I Spring 1993

[Bac04] M. Bachler, S. Buckingham Shum, J. Chen-Burger, J. Dalton, D. De Roure, M. Eisenstadt, J. Komzak, D. Michaelides, K. Page, S. Potter, N. Shadbolt, and A. Tate, "Collaborative Tools in the Semantic Grid," The Eleventh Global Grid Forum, Honolulu, Hawaii, USA, 2004.

[Bac93] T. Bäck and H. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization", Evolutionary Computation, Vol. I Issue I Spring 1993

[Bak05] M. Baker and G. Smith, “GridRM: A Resource Monitoring Architecture for the Grid”. 3rd International Workshop on Grid Computing, Baltimore, Maryland, USA, November 2005.

[Bal01] Z. Balaton, P. Kacsuk, and N. Podhorszki, “Application Monitoring in the Grid with GRM and PROVE”, International Conference on Computational Science - ICCS 2001, San Francisco, CA., USA (2001).

[Bal03] Z. Balaton and G. Gombas. “Resource and Job Monitoring in the Grid”. Euro-Par 2003 International Conference on Parallel and Distributed Computing, Klagenfurt, Austria (ICPDC 2003).

[Bav04] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. “Operating System Support for Planetary-Scale Network Services”. 1st Networked Systems Design and Implementation (NSDI ‘04), May 2004

[Ber97] F. Berman and R. Wolski. “The AppLeS Project: A Status Report”. Proceedings of the 8th NEC Research Symposium, Berlin, Germany, May 1997.

[Bio04] A. S. Grimshaw, G. Robins, M. Humphrey, and M. Morgan. "The Global Bio Grid at Virginia", Computer Science Technical Report, University of Virginia, CS-2004-27, September, 2004.

[Bor05] Borja Sotomayor and Lisa Childers. “Globus Toolkit 4: Programming Java Services”. The Elsevier Series in Grid Computing by Morgan Kaufmann” 1st Edition (December 2005).

[Buy99] R. Buyya. “High Performance Cluster Computing: Architectures and Systems”. Volumes 1 and 2, Prentice Hall, NJ, USA, 1999.

[Buy00] R. Buyya, D. Abramson, and J. Giddy. “Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid”. The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), May 2000, Beijing, China, IEEE Computer Society Press, USA.

[Buy01a] R. Buyya, K. Branson, J. Giddy, and D. Abramson. “The Virtual Laboratory: Enabling Molecular Modelling for Drug Design on the World Wide Grid”. Technical Report, Monash-CSSE-2001-103, Monash University, Melbourne, Australia, December 2001.

[Buy01b] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. “Economic Models for Management of Resources in Peer-to-Peer and Grid Computing” In Proceedings of International Conference on Commercial Applications for High-Performance Computing, SPIE Press, August 20-24, 2001, Denver, Colorado, USA.

[Buy02] Rajkumar Buyya. “Economic-based Distributed Resource Management and Scheduling for Grid Computing”. Ph.D. Thesis, Monash University, Melbourne, Australia, April 12, 2002.

C

[Cab07] A. Cabani, S. Ramaswamy, M. Itmi, S. Al-Shukri and J.P. Pécuchet. “Distributed Computing Systems: P2P versus Grid Computing Alternatives Innovations and Advanced” Techniques in Computer and Information Sciences and Engineering” September 04, 2007.

[Cas97] H. Casanova and J. Dongarra. “NetSolve: A Network Server for Solving Computational Science Problems”. International Journal of Supercomputing Applications and High Performance Computing, Vol. 11, No. 3, pp 212-223, Sage Publications, USA, 1997.

[Cas07] The GridCast Project Web Site http://gridcast.web.cern.ch/gridcast/

[Cap05] F. Cappello, et al., “Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform”, 6th IEEE/ACM Grid Computing, Grid'2005, Nov 13-14, 2005, Seattle, Washington, USA.

[Cha99] S. Chapin, J. Karpovich, and A. Grimshaw. “The Legion Resource Management System, Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing”. April 16, 1999, San Juan, Puerto Rico, USA, Springer Verlag Press, Germany, 1999.

[Cha00] Steven C. Chapra and R. Canale, “Numerical Methods for Engineers: With Software and Programming Applications”. McGraw-Hill ISBN: 0072431938

[Cha05] M. Chaudier, J. Gelas and L. Lefèvre, “Towards the design of an autonomic network node”, International Working Conference on Active and Programmable Networks, Nice, France, Nov. 21-23, 2005.

[Che00] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, an S. Tuecke. “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets”. Journal of Network and Computer Applications, Vol. 23, No. 3, July 2000.

[Chen02] H. Chen and M. Maheswaran. “Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems”. In Proc. of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), pp. 88--97, Fort Lauderdale, Florida USA.

[Che05] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, B. Moe. “Wide Area Data Replication for Scientific Collaborations”. Proceedings of 6th IEEE/ACM International Workshop on Grid Computing (Grid2005), November 2005.

[Cis06] Cisco Systems, Inc. “The Cisco IOS IP Service Level Agreements – White Paper”. September, 2006.

[Cza98] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. “A Resource Management Architecture for Metacomputing Systems”. In The 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998

[Cza01] S. Czajkowski, K. Fitzgerald, I. Foster, C. Kesselman, Grid Information Services for Distributed Resource Sharing, in: Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Computer Society Press, San Francisco, CA, 2001, pp. 181–194.

[Cza06] K. Czajkowski, et al., “The WS-Resource Framework”. The OASIS Organization. Version 1.2 April 2006.

D

[Dat07] European DataGrid http://eu-datagrid.web.cern.ch

[Dew04] A. DeWitt, T. Gross, B. Lowekamp, et al., "ReMoS: A Resource Monitoring System for Network-Aware Applications". Carnegie Mellon School of Computer Science.

[DMTF] Distributed Management Task Force Web Site: http://www.dmtf.org/

[Doe07] DOE Science Grid http://www.doesciencegrid.org/

[Don06] F. Dong and Selim G. Akl. “Scheduling Algorithms for Grid Computing: State of the Art and Open Problems”. Technical Report No. 2006-504 School of Computing, Queen’s University Kingston, Ontario January 2006

E

[Ege07] The Enabling Grids E-SciencE (EGEE) Project Web Site: http://public.eu-egee.org/

F

[Fan06] A. Fangpeng and Selim Akl. “Technical Report No. 2006-504 Scheduling Algorithms for Grid Computing: State of the Art and Open Problems”. January 2006

[Fie93] Scott Fields, "Hunting for Wasted Computing Power", 1993 Research Sampler, University of Wisconsin-Madison Virtual Organization”. International Journal of Supercomputer Applications, vol. 15 no. 3, Sage Publications, USA 2001.

[Fis80] M.L. Fisher, "Worst-case Analysis of Heuristic Algorithms", Management Science Vol. 26 Issue I (1980)

[Fos99] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. “A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation”. In the International Workshop on Quality of Service, June 1999.

[Fos00] I. Foster and Carl Kesselman. “Computational Grids”. Morgan-Kaufman, 2000

[Fos01] I. Foster, C. Kesselman and S. Tuecke, “The Anatomy of the Grid: Enabling Scalable” Technical Report, 2001.

[Fos02a] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “The Physiology of the Grid: An Open Services Architecture for Distributed Systems Integration” Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[Fos02b] Foster, I. “What Is the Grid? A Three Point Checklist”. Grid Today, 1(6), 2002.

[Fos03] I. Foster and C. Kesselman, eds., “The Grid2: Blueprint for a Future Computing Infrastructure”, Morgan Kaufmann, San Francisco, 2003

G

[Gar00] A. Garrido, M.A. Salido and F. Barber. “Heuristic Methods for Solving Job-Shop Scheduling Problems”. ECAI-2000 Workshop on New Results in Planning, Scheduling and Design. Berlín Pp. 36-43. 2000.

[GGF07] Global Grid Forum. Web Site: http://www.ggf.org

[Gli07] The gLite Project Web Site: http://glite.web.cern.ch/glite/

[Glo93] F. Glover, and M. Laguna, "Tabu Search", John Wiley & Sons, Inc. 1993

[Glo07a] Globus Resource Allocation Manager (GRAM). http://www.globus.org/gram/.

[Glo07b] Global Grid Forum Scheduling and Resource Management Area (SRM). http://www.mcs.anl.gov/~jms/ggf-sched.

[Glo07c] GGF Distributed Resource Management Application API Working Group (DRMAA-WG). http://www.drmaa.org/.

[Gon06] A. Gonzalez Prieto and R. Stadler, "Adaptive Distributed Monitoring with Accuracy Objectives," ACM SIGCOMM workshop on Internet Network Management (INM 06), Pisa, Italy, September 11, 2006.

[Gra02] S. Graupner, V. Kotov, A. Andrzejak and H. Trinks, “Control Architecture for Service Grids in a Federation of Utility Data Centers”, HP Labs., Palo Alto, USA. August 2002.

[Gra07a] GGF Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG). http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html.

[Gra07b] GGF Scheduling Disctionary Working Group (SD-WG).

http://www.fz-juelich.de/zam/RD/coop/ggf/sd-wg.html, 2003.

[Gri5000] The Grid'5000 Web page: https://www.grid5000.fr/

[GridLab] GridLab. A Grid Application Toolkit and Testbed. http://www.gridlab.org/

[Gro01] William Grosso. “Java RMI” O’Reilly Editorial qOctober 01, 2001 ISBN-10: 1-565-92452-5

H

[Hol75] Holland, John H. “Adaptation in Natural and Artificial Systems”. University of Michigan Press, Ann Arbor

I

[Iam03] A. Iamnitchi and I Foster. “On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing” In: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03).

[IETF] IETF Policy Web Site: http://www.ietf.org/html.charters/policy-charter.html

[ITU00] ITU-T Rec E.800: Terms and definitions related to quality of service and network performance including dependability: http://www.itu.int/rec/T-REC-E.800-199408-I/

J

[Jac06] Bart Jacob. “Grid computing: What are the key components?”. ITSO Redbooks Project Leader, IBM Jun 2006

[Jan07] Sung Ho Jang and Jong Sik Lee Service. “Agent-Based Resource Management Using Virtualization for Computational Grid”. Computational Science ICCS 2007 Baijing, China.

[Jia06] Jia Yu and Rajkumar Buyya, “A Taxonomy of Workflow Management Systems for Grid Computing” Grid Computing and Distributed Systems (GRIDS) Laboratory. The University of Melbourne, Australia http://www.gridbus.org

[JAPRO] Java Profiler Toolkit 6.0: http://java.sun.com/

K

[KAD] The KADEPLOY 2 Toolkit. Web Page: http://kadeploy.imag.fr/

[Kap99] N. Kapadia and J. Fortes. “PUNCH: An Architecture for Web-Enabled Wide-Area Network Computing”. Cluster Computing: The Journal of Networks, Software Tools and

Applications, Vol. 2, No. 2, pp. 153-164, Baltzer Science Publishers, The Netherlands, Sept. 1999.

[Kel05] Noel Kelly, P.V. Jithesh, Sachin Wasnik, Roy McLaughlin, Fabiola Fragoso, Paul Donachy, Terence Harmer, Ron Perrott, Mark McCurley, Michael Townsley, Jim Johnston, Shane McKee, "The GeneGrid Portal: A User Interface for a Virtual Bioinformatics Laboratory". UK e-Science All Hands Meeting (AHM 2005), Nottingham, Sep 19-22, 2005

[Kir83] S. Kirkpatrick, C. Gelatt Jr., M. Vecchi, "Optimization by Simulated Annealing", Science 220, ISSN 671-680, 1983.

[Kis05] T. Kiss, G. Terstyanszky, G. Kecskemeti, Sz. Illes, T. Delaittre, S. Winter, P. Kacsuk, G. Sipos. ”Legacy Code Support for Production Grids”. Proc. of the Grid 2005 - 6th IEEE/ACM International Workshop on Grid Computing November 13-14, 2005, Seattle, Washington, USA

[Kli04] T. Klie and F. Strauß. Integrating SNMP agents with xml-based management systems. IEEE Communications, 42(7):76–83, July 2004.

[Kra02] K. Krauter, R. Buyya, and M. Maheswaran. „A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing”. International Journal of Software: Practice and Experience (SPE), Wiley Press, New York, USA, May 2002.

[Kur04] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak and J. Pukacki. “Improving Grid Level Throughput Using Job Migration And Rescheduling”. Scientific Programming vol.12, No.4, pp. 263-273, 2004.

L

[Law85] E. Lawler, J. Lenstra, A. Rinnooy and D. Shmoys, "The Travelling Salesman: A Guided Tour of Combinatorial Optimization", John Wiley and Sons. Edit. 1985 ISBN: 0471904139

[Law00] JAMM Project. “Java Agents for Monitoring and Management”. Lawrence Berkeley National Laboratory. http://www-didc.lbl.gov/JAMM/. July, 2000.

[Ldap07] The LDAP RFC: http://tools.ietf.org/html/rfc4510

[Leg04] I. Legrand, H. Newman, et al., “MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications” CHEP 2004, Interlaken, Switzerland, September 2004.

[Lhc07] The LHC Project Web Site: http://lcg.web.cern.ch/LCG/

M

[Maga04] E. Magaña, E. Salamanca and J. Serrat, “Proposal of a Policy-Based System for Grid Services Management”, APGAC'04/ICCS'04 Krakow, Poland June 2004

[Maga05] E. Magaña and J. Serrat, "QoS Aware Resource Management Architecture for OGSA Services Deployment", 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005), 15-19 May. Nice - Acropolis, France. (Poster)

[Maga06] E. Magaña. Grid5000 Results: http://nmg.upc.es/~emagana/sblomars/grid5000.html

[Maga07a] E. Magaña, L. Lefevre and J. Serrat. “Autonomic Management Architecture for Flexible Grid Services Deployment Based on Policies”. ARCS'07, Zurich, Switzerland. 2007.

[Maga07b] E. Magaña, L. Lefevre, and J. Serrat. “SBLOMARS: SNMP-based Balanced Load Monitoring Agents for Resource Scheduling in Large-scale Grids”. Las Vegas, Nevada. USA 2007

[Maga07c] E. Magana, M. Hasan and J. Serrat. “Heuristic Grid Resource Management System” ICNS 2007. June 19-25, 2007 - Athens, Greece.

[Maga07d] E. Magaña, L. Lefevre, J. Serrat, “SNMP-based Monitoring Agents and Heuristic Scheduling for large scale Grids”. GADA07 : Grid Computing, high-performAnce and Distrributed Applications, Portugal, November 2007.

[Maga07e] E. Magana, Joan Serrat, Laurent Lefevre and Masum Hasan. "Service Oriented Architecture for Large-Scale Grids", Computer Communications Journal. Elsevier Editorials 2008 (Submitted).

[Mas04] M. Massie, B. Chun, and D. Culler. “The Ganglia Distributed Monitoring System: Design, Implementation, and Experience”. Parallel Computing, Vol. 30, Issue 7, July 2004.

[Mec99] C. Mechoso, C. Ma, J. Farrara, J. Spahr, and R. Moore. “Parallelization and distribution of a coupled atmosphere-ocean general circulation model”. Mon. Rev., 121:2062, 1999.

[Mes06] K. Mesghouni, S. Hammadi, P. Borne. “Evolutionary Algorithms for Job-Shop Scheduling. Ecole Centrale de Lille, LAIL - UMR 8021 BP 59651 Villeneuve d’Ascq Cedex, France.

[Min04] Hwa Min Lee; Sung Ho Chin; Jong Hyuk Lee; Dae Won Lee; Kwang Sik Chung; Soon Young Jung; Heon Chang Yu. ”A Resource Manager for Optimal Resource Selection And Fault Tolerance Service In Grids”. Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on Volume, Issue , 19-22 April 2004

N

[Nab04] J. Nabrzyski, J. M. Schopf and J. Weglarz, “Grid Resource Management State of the Art and Future Trends” Kluwer Academic Publishers. Boston, USA October 2004.

[Nad06] Nadiminti, Dias de Assunção, Buyya. "Distributed Systems and Recent Innovations: Challenges and Benefits". InfoNet Magazine, Volume 16, Issue 3, Melbourne, Australia.

[Nak99] H. Nakada, M. Sato, S. Sekiguchi. “Design and Implementations of Ninf: towards a Global Computing Infrastructure”. Future Generation Computing Systems, Metacomputing Special Issue, October 1999.

[Nea00] M. Neary, A. Phipps, S. Richman, P. Cappello. “Javelin 2.0: Java-Based Parallel Computing on the Internet”. Proceedings of European Parallel Computing Conference (Euro-Par 2000), Germany, 2000.

[NGG06] Next Generation GRIDs Expert Group Report 3. “Future for European Grids: GRIDs and Service Oriented Knowledge Utilities“ January 2006

http://cordis.europa.eu/ist/grids/ngg.htm

[Nic98] K. Nichols, S. Blake. "Differentiated Services Operational Model andDefinitions", Work in Progress, draft-nichols-dsopdef-00.txt, February, 1998

[Nie96] J. Nieplocha and R. Harrison. “Shared memory NUMA programming on the I-WAY”. In Proc. 5th IEEE Symp. on High Performance Distributed Computing. IEEE Computer Society Press, 1996.

[Nor96] M. Norman, P. Beckman, G. Bryan, J. Dubinski, D. Gannon, L. Hernquist, K. Keahey, J. Ostriker, J. Shalf, J. Welling, and S. Yang. “Galaxies Collide On The I-WAY: An Example Of Heterogeneous Wide-Area Collaborative Supercomputing”. International Journal of Supercomputer Applications, 10(2):131{140, 1996.

[Nor07] The NorduGrid Project Web Page, http://www.nordugrid.org/

O

[OAR] The OAR 2 Toolkit. Web Page: http://oar.imag.fr/

[OGF] The Open Grid Forum Web Site: http://www.ogf.org/

[Orf00] Robert Orfali, Dan Harkey, Jeri Edwards. “The Essential Distributed Objects Survival Guide”. John Wiley & Sons, ISBN 0-471-12993-3

[Osg07] The Open Science Grid Project Web Site: http://www.opensciencegrid.org/

P

[Pag05] A. Page and T. Naughton. “Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing”. 19th IEEE IPDPS. April 2005. Colorado. USA.

[Pav04] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta. “On management technologies and the potential of web services”. IEEE Communications Magazine, 42(7):58–66, July 2004.

[Pra04] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, “Comparing the performance of SNMP and web services based management”. IEEE Electronic Transactions on Network and Service Management, 2004.

R

[Ram02] Michael Rambadt, Philipp Wieder. “UNICORE - Globus: Interoperability of Grid Infrastructures”. Conf. Proc of the Cray User Group Summit 2002, Manchester, UK.

[Ram03] N. Ramakrishnan, L. T. Watson, D. G. Kafura, C. J. Ribbens, and C. A. Shafer. “Programming Environments for Multidisciplinary Grid Communities”. Concurrency and Computation: Practice and Experience, 2003

[Ram05] Ramesh Subramanian and Brian Goodman. “Peer-to-Peer Computing: Evolution of a Disruptive Technology”. ISBN 1-59140-429-0, Idea Group Inc., Hershey, PA, USA, 2005.

[RC5] The RC5 Project:http://www.distributed.net/rc5/

[Rev95] C. Reeves, "Modern Heuristic Techniques for Combinatorial Problems", McGraw-Hill Book Company, UK 1995. ISBN 07-709239-2.

[Ric07] A. Ricadela, “Computing Heads for the Clouds”. BusinessWeek Internet Article. http://www.businessweek.com/technology/content/nov2007/

[Rus06] Elliotte Rusty Harold. “Java Network Programming”, 3rd Edition O'Reilly Editorial Group ISBN: 0596007213 New Edition 2006

S

[Sak04] Rizos Sakellariou, Henan Zhao. “A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems”. Heterogeneous Computing Workshop, IEEE Computer Society Press, 2004.

[Sal03] Salamanca, E., E. Magaña, J. Vivero, A. Galis, B. Mathieu, Y. Carlinet, O. Koufopavlou, C. Tsarouchis, C. Kitahara, S Denazis and J. L. Mañas: "A Policy-Based Management Architecture for Active and Programmable Networks" IEEE Network Magazine, Special issue on Network Management of Multiservice Multi-media IP-Based Networks, May 2003, Vol. 17 No.3

[Sch03] J. Schönwälder, A. Pras, and J. P. Martin-Flatin. “On the Future of Internet Management Technologies”. IEEE Communications Magazine, 41(10):90–97, October 2003

[SETI] The SETI@Home Project: http://setiathome.ssl.berkeley.edu/

[SNINF] SNMP Informant Windows Agents: http://www.wtcs.org/informant/

[Sta99] W. Stallings, “SNMP, SNMPv2, SNMPv3 and RMON 1 and 2, (Third Edition)”. Addison-Wesley Professional, pages 365 - 398. 1999.

[Stra03] J. Strassner. “Policy-Based Network Management: Solutions for the Next Generation”. The Morgan Kaufmann Series in Networking. USA 2003. Pag. 503

[Sub00] R. Subramanyan, J. M. Alonso, J. Fortes. “A scalable SNMP-based distributed monitoring system for heterogeneous network computing.” SC2000 in November 4 – 10, Dallas Convention Center, 650 South Griffin Street Dallas, TX.

T

[Tan01] Andrew S. Tanenbaum and Maarten Van Steen. “Distributed Systems: Principles and Paradigms”. Prentice-Hall, September 2001.

[Ter07] The TeraGrid Project Web Site: http://www.teragrid.org/

[Tha05] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[Tho05] Erl, Thomas (2005). "Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River". Prentice Hall PTR. ISBN 0-13-185858-0.

[Tie02] B. Tierney and D. Gunter, "NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging". LBNL Tech Report LBNL-51276, 2002.

[Tie03] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski, A Grid Monitoring Architecture, GWDPerf-16–3, Global Grid Forum, August 2003. http://wwwdidc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf.

[TMN00] Specification documents for TMN and FCAPS (M.3000, M.3100, M.3200, M.3400). http://www.itu.int

U

[UCDSN] UCD-SNMP: http://net-snmp.sourceforge.net/docs/mibs/ucdavis.html

[Uke07] UK's e-Science program http://www.rcuk.ac.uk/escience/default.htm

V

[Ved02] Vedamuthu, et al., "Web Services Policy 1.5 – Framework". W3C Working Draft.

[Ven06] Venters,W., Cornford,T. "Introducing Pegasus: An Ethnographic Research Project Studying the Use of Grid Technologies by The UK Particle Physics Community". Second International Conference on e-Social Science, 28-30 June 2006, Manchester, UK.

[Ver01] D. Verma, "Policy-Based Networking: Architecture and Algorithms", New Riders Publishing. California, USA 2001. ISBN:1578702267

W

[Wee04] Alex Weeks. "1.1", Linux System Administrator's Guide, version 0.9. USA 2004.

X

[XML] The eXtensible Markup Language Web Page: http://www.w3.org/XML/

Y

[Yad07] Rajinder Yadav. "Client / Server Programming with TCP/IP Sockets" Sept 9, 2007 Reference Web Page: http://devmentor.org

[Yan02] K. Yang, A. Galis, C. Todd. "Policy-Based Active Grid Management Architecture". 10th IEEE International Conference on Networks ICON 2002.

Z

[Zan05] S. Zanikolas and R. Sakellariou, "A Taxonomy of Grid Monitoring Systems". FGCS Journal. January 2005

[Zom01] A. Zomaya and Y. H. The, “Observations on using genetic algorithms for dynamic load-balancing”. IEEE Transactions on Parallel and Distributed Systems, 12 (9):899-911, September 2001.