Dipartimento di Informatica e
Scienze dell'Informazione

An architectural approach to the management of applications with QoS requirements on Grid

by

Alessio Merlo

Theses Series **DISI-TH-2010-XX**

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e
Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

An architectural approach to the management of applications with QoS requirements on Grid

by

Alessio Merlo

July, 2010

Dottorato di Ricerca in Informatica
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Università degli Studi di Genova
Via Dodecaneso 35
16146 Genova, Italy
http://www.disi.unige.it/

Ph.D. Thesis in Computer Science (S.S.D. INF/01)

Submitted by Alessio Merlo
DISI, Università degli Studi di Genova
merlo@disi.unige.it

Date of submission: 17th of December, 2009

Title: An architectural approach to the management of applications with QoS requirements on Grid

Advisor: Vittoria Gianuzzi
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova
gianuzzi@disi.unige.it

Advisor: Andrea Clematis
Istituto di Matematica Applicata e Tecnologie Informatiche (IMATI)
Consiglio Nazionale delle Ricerche (CNR), Genova
clematis@ge.imati.cnr.it

Advisor: Angelo Corana
Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT)
Consiglio Nazionale delle Ricerche (CNR), Genova
corana@ieiit.cnr.it

Abstract

Grid Computing is surely one of the most interesting and widely studied distributed paradigms of the last decade. During years of research, development and test, Grid has evolved in different directions, far away from its original conception.

In particular, Grid Computing has been originally designed as an universal platform to share and access world-wide resources in order to support the management and execution of large scientific jobs. Despite the first academic target, the potentiality of Grid as an architecture for sharing any kind of resources had made the platform suitable for other kinds of tasks.

For this, Grid evolved from scientific to commercial purposes, extending the number of potential applications. From a system perspective, this corresponds to an increase in the kinds of jobs to manage by the Grid and a raise of the QoS level that the Grid users demand.

Under this perspective, it is important to find out how the new Grid applications can be efficiently managed over a platform that was not designed for their management in its original conception. In particular, the contribution of this thesis is dedicated to the study of jobs with time constraints on execution. Currently, these jobs are not efficiently managed by any Grid infrastructure.

Because this topic is general enough, it has been studied from different perspectives, in order to make the research activity consistent and a good basis for further works. For this, the aim is to provide a proper classification of the problem and a contribution both at the theoretical and at the implementation levels.

To this regard, this thesis is divided into three main parts. At first, a deep analysis of the state of the art regarding the management of QoS constraints is made. The aim of such analysis is double: in primis, it is important for showing the inadequacy of current Grid framework solutions for supporting time constraints on Grid; moreover, it allows to justify the need of time constraints management on Grid through an evaluation of the evolution of the Grid under a QoS perspective.

The second contribution of the thesis is the definition of a proper Grid framework architecture, SoRTGrid, for supporting the management of time constraints on Grid. SoRTGrid framework has been defined and designed starting from the analysis of the state of the art carried out in the first part. In particular, the architectural choices have been made

considering the lacks of the current QoS framework proposals. In practical, the SoRTGrid framework realizes an overlay structure that extends the Grid middleware functionalities providing both the explicit management of time constraints and an optimized platform for supporting direct interactions of autonomous Owner and User Agents for an active resource market. In this sense, SoRTGrid proposes a new approach to resource discovery and acquisition in comparison with the resource management approaches of a typical Grid middleware.

Because the SoRTGrid overlay is made to work on different general-purpose Grid middleware, it has been implemented in Java and XML for the sake of universality, in order to grant a simple portability on such middleware. For this, after the completion of the general prototype implementation, the research activity has been divided in two parallel tasks: (1) the porting of the SoRTGrid prototype on a simulated Grid, and (2) an experimental evaluation of the architectural schema proposed in SoRTGrid.

Regarding the porting activity, SoRTGrid prototype has been currently deployed on a Grid simulated by the GridSim Toolkit [BMA02]. Such step has been realized with two aims: firstly, the need to prove that the implemented architecture could effectively be ported and executed on a Grid structure and, secondly, the need to provide a working application for testing both different behaviors for the independent agents composing SoRTGrids and algorithms for the services offered by the framework (e.g. distributed discovery). The testing of this first porting activity is currently still in progress and is out of the scope of this thesis.

The evaluation of the SoRTGrid architecture is the third and last contribution of this thesis. This activity regards the verification that the proposed overlay architecture works well on the variable and heterogeneous Grid scenario. Such evaluation has to be made on a very abstract Grid scenario (without latencies or issues related with the Grid infrastructure or middleware), so the use of a (logical or real) Grid or the current SoRTGrid prototype is excluded. Therefore, the evaluation activity has been carried out through the implementation of a proper "out-of-grid" simulator called SoRTSim (a finite-event parallel simulator implemented in Java) that simulates the SoRTGrid overlay on different computational scenarios, represented at a high abstraction level. In this sense, SoRTSim has been used to test the efficiency of its structure on different underlying scenarios, from simple to complex ones (referable to a Grid), in order to demonstrate the adequacy of the proposed architecture on a Grid context. The current implementation of SoRTSim is general and modular; for this, it can be used out of the SoRTGrid scope, for simulating other distributed overlay architectures (e.g. P2P).

To Mum and Dad,
To Tata

Acknowledgements

Contents

Chapter 1

Grid, Real-Time and QoS: state of art and open issues

1.1 General aspects

The Grid Computing paradigm wants to realize a common distributed system in which every resource could be shared and used by any user around the world independently of the organization the user and the resource belong to.

The final aim of such a new-coming paradigm can be easily understood: every resource is not a part of a single organization but can be shared, together with many other resources from different domains, into a common platform accessible from all users around the world. The potential of this approach is likewise clear: all resources in a unique, big and worldwide distributed system constitutes a huge repository of computational, storing and network power that can be accessed to retrieve enough amount of resources to satisfy any user request. The resources in the "global" repository could be used to solve computational intensive problems like, for instance, Grand Challenge problems (e.g. protein folding [ea06], financial modeling [WC04], earthquake simulation[ea02a] [nee], and climate/weather modeling [ea05]), if sufficient Grid Computing resources are provided.

Previous challenges need huge amount of resources, beyond the capabilities of a single organization. For this, the initial aim of Grid Computing has been the sharing of unused resources for managing computing intensive Best Effort (BE) jobs. Notwithstanding the original purpose, during its evolution the nature of Grid as a global repository of resources made it interesting even for other kinds of applications. The need of managing such new applications has evolved the Grid from a best effort platform to a new one able to provide different levels of Quality of Service (QoS).

In this thesis, I deeply investigate QoS issues on Grid Computing, with a dual aim: firstly, I want to evaluate the effective QoS support on Grid at present; secondly, I aim to study new architectural solutions for extending the kind of QoS applications currently managed on Grid towards applications with time constraints on jobs termination.

1.1.1 Thesis outline

In this chapter, I firstly provide a general introduction to Grid Computing and Real Time paradigms. From this, an extensive state-of-the-art on the evolution of the management of QoS on Grid is provided, in order to highlight the current open issues in this field. In particular, the limitation of the current support to QoS is shown, as the lack of global and universal solutions. In the final part of the chapter, a set of potential Grid applications with strict QoS constraints (i.e. time-constrained jobs) -not currently supported on Grid- are introduced. The efficient management of such kind of QoS applications is the main aim of the thesis.

Chapter 2 introduces SoRTGrid, a QoS framework able to provide support also for time-constrained applications. In particular, I provide an abstract model for soft-real time jobs and I show how the architectural choices made in SoRTGrid are able to support time-constraints on jobs execution, both on a classical and an Economic Grid.

Chapter 3 is devoted to the description of the implementation of SoRTGrid in Java. In particular, technologies and choices made to simplify the management of user side requirements and owner side offers are detailed and justified. Finally, since SoRTGrid can work on different kinds of Grid (and middleware), a strategy for porting the framework on a simulated Grid (i.e. GridSim) is presented.

Chapter 4 investigates a methodology for evaluating the efficiency of SoRTGrid on a real context. Since it is impossible to evaluate the SoRTGrid prototype on a real Grid (i.e. the current implementation is not sufficient to port it on effective Grid scenarios), this chapter consider different techniques (simulation, emulation and in-situ) to evaluate the efficiency of the SoRTGrid solution. In particular, the simulation technique is chosen as the best one. To this regard, a justification of the choice made and an analysis of set of Grid simulators are given.

Chapter 5 introduces SoRTSim. Since the evaluation of Grid simulators in the previous chapter has shown that a satisfying solution for the evaluation of SoRTGrid framework is not available, a dedicated simulation tool for SoRTGrid (i.e. SoRTSim) has been designed and implemented. This chapter describes the SoRTSim architecture and implementation, and shows the porting of SoRTGrid structures to the SoRTSim simulator.

Chapter 6 presents the results of several experiments made with SoRTSim for simulating

the behavior of SoRTGrid in different Grid scenarios. In particular, eight environments have been analyzed and tested, from simple distributed systems to complex economic grids. The obtained results are critically analyzed and a global evaluation of the SoRTGrid behavior is provided.

Chapter 7 concludes the thesis by defining some possible extensions and future works. In particular, two kinds of potential extensions are proposed: the first set regards SoRTSim and the second one is related to the SoRTGrid framework.

1.1.2 Projects and publications

This thesis was supported in part by the regional programme of innovative actions of the European fund PRAI-FESR Liguria.

Most of the results of the thesis are the subject of the following publications.

A critical analysis of the current state of art of QoS on Grid Compunting is the subject of:

- A. Merlo, A. Clematis, A. Corana, V. Gianuzzi
  Quality of Service on Grid: architectural and methodological issues
  Concurrency and Computation: Practice and Experience, in print.

SoRTGrid have been described on:

- A. Merlo, A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi, A. Quarati
  SoRTGrid: a Grid framework compliant with Soft Real Time requirements
  in: 'Remote Instrumentation and Virtual Laboratories: Service Architecture and Networking', F. Davoli, N. Meyer, R. Pugliese, S. Zappatore (Eds.), (Springer, Berlin, 2010) ISBN 978-1-4419-5595-1, pp. 145-161.

SoRTSim have been described in:

- A. Merlo, A. Corana, V. Gianuzzi, A. Clematis
  SoRTSim: A high-level simulator for the evaluation of QoS models on Grid,
  accepted paper at '5th International Workshop on Distributed Cooperative Laboratories: Instrumenting the Grid' (INGRID 2010), May 2010, Poznan, Poland.

Other works related with the technologies adopted in the SoRTGrid framework are:

- A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi, A. Merlo
  Resource selection and application execution in a Grid: A migration experience from GT2 to GT4
  Lecture Notes in Computer Science n. 4276, 'On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA and ODBASE', R. Meersmann, Z. Tari et al. (Eds.), (Springer, Berlin, 2006) pp. 1132-1142.

- A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi, A. Merlo
  Grid services for 3D data analysis in Virtual Laboratories
  in 'Grid Enabled Remote Instrumentation', Series: Signals and Communication Technology,
  F. Davoli, N. Meyer, R. Pugliese, S. Zappatore (Eds.), (Springer, Berlin, 2008) ISBN 978-0-387-09662-9, pp. 481-498.

- A. Merlo, D. DAgostino, V. Gianuzzi, A. Clematis, A. Corana
  GridWalker: a visual tool for supporting the advanced discovery of Grid resources
  Proc. 4th Int. Workshop on Distributed Cooperative Laboratories: Instrumenting the Grid (INGRID 2009), Alghero (Italy), 2009, Springer, pre-print.

- A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi, A. Merlo, A. Quarati
  A distributed Approach for structured Resource Discovery on Grid
  Proc. 2nd Int. Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008, Barcelona), (IEEE Computer Society, 2008) ISBN 0-7695-3109-1, pp. 117-125.

- V. Gianuzzi, A. Merlo, A. Clematis, D. DAgostino
  Managing Networks of Mobile Entities using the HyVonNe P2P Architecture
  Proc. 2nd Int. Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008, Barcelona), Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC), (IEEE Computer Society, 2008) ISBN 0-7695-3109-1, pp. 335-341.

- V. Gianuzzi, D. D'Agostino, A. Merlo, A. Clematis
  Efficient management of resources and entities using the HyVonNe P2P architecture
  International Journal of Grid and Utility Computing (IJGUC), Special Issue on Efficient Techniques for Resource and Service Management in Grid and P2P-based Applications, v. 1, n. 3, 2009, pp. 216-226.

- A. Merlo
  The Condor on the Grid: State of Art and Open Issues
  First Java To Grid Middleware (J2GM) Workshop, DIST Università di Genova, February 2007.

1.2 Introducing Grid basic concepts

The *global infrastructure*[FK03] of Grid Computing is based on three concepts, i.e. the Grid user, the resources and the Virtual Organization.

A Grid user is defined as "everyone who can need resources". It can be any entity (single user, research group, etc) that requires Grid resources to run a given job or application. On the other side, a Grid resource is "everything that could be shared": it can be physical (e.g. CPU, RAM, disk space) but also logical (e.g. a distributed File System or a software license): what is fundamental for a resource to be a Grid resource is that it is available through an appropriate and well-defined interface.

To obtain such result, a first need is that both users and resources could be identified universally, out of the single definition and the standards of the administrative domain. For this, Grid Computing paradigm is based on the concept of Virtual Organizations (VOs).

By definition, a Virtual Organization is composed by:

- A set of virtualized resources;
- A set of Grid users;
- A set of policies that define the rules through which virtual resources can be accessed.

Virtualization, sharing and usage of virtual resources by virtual users through universally recognized rules, is the main aim of Grid Computing. To obtain such result in practice, Grid Computing is implemented through an appropriate middleware that is commonly shared by the physical hosts composing the Grid. The main requirement of a Grid middleware is to manage a considerable suite of open standards and mechanisms granting a real virtualization over the proprietary protocols adopted by the single organizations, for an efficient interaction between users and resources.

Grid middleware implements Grid Computing specifications, making the Grid an effective working platform.

Current Grids are the results of years of standardization and implementation activities. Such evolutionary process of the Grid has been guided by different factors that changed significantly the aims and the way to conceive both Virtual Organizations and the physiology of the Grid. Moreover, the availability of a huge amount of potential resources made the Grid suitable for many classes of applications.

From an historical point of view, in its first conception the Grid has been uniquely defined as a platform for sharing resources. For this reason, any Grid component, from the layered architecture definition [IFT01] to the first implementation of a full working middleware

Figure 1 A pattern of extra-grid virtual organizations

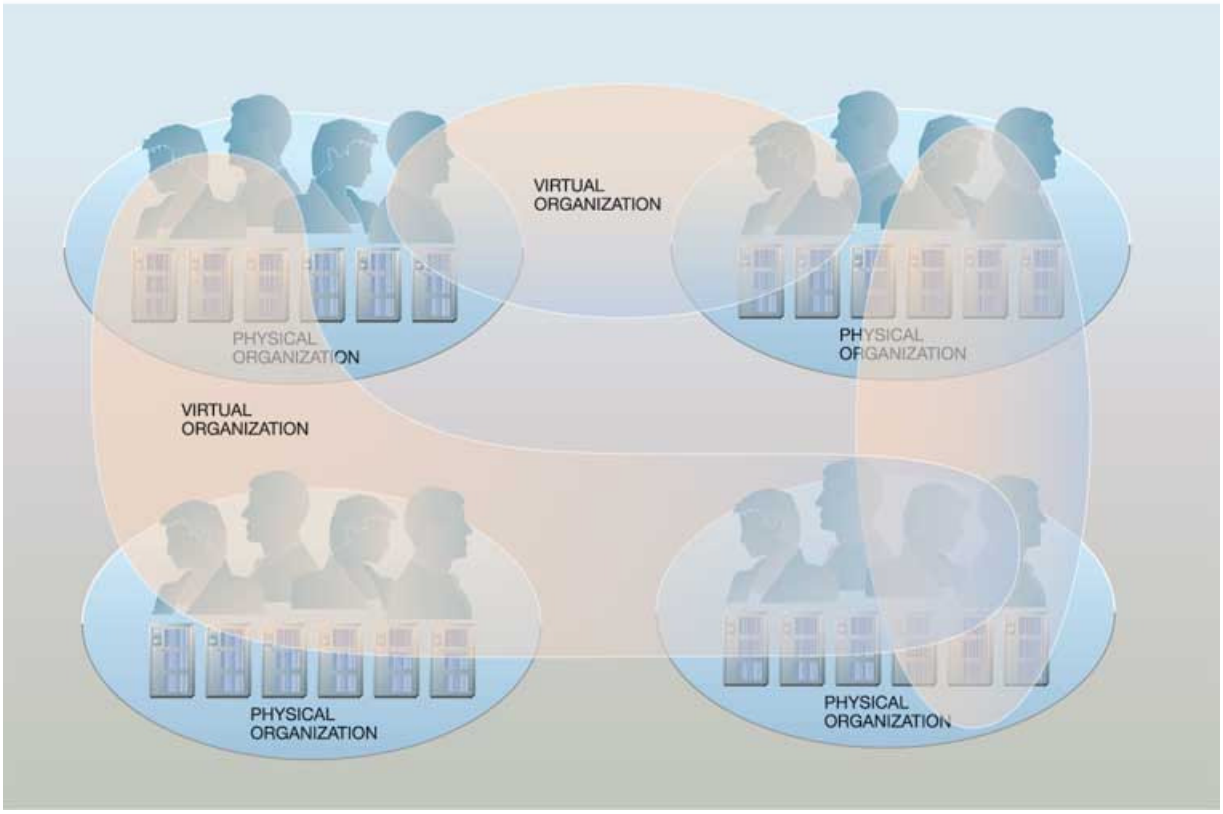

Figure 1.1: The idea of Virtual Organization

[FK98] has been deployed with the unique aim to provide a real universal and common platform to share resources beyond the limitation of the single administrative domain, with no regard to the level of efficiency in matchmaking jobs and resources.
More specifically, the focus of early Grid Computing has been the definition of a proper abstraction level (i.e. Virtual Organization) to share resources, while the management of such resources had a secondary importance.
As a consequence, discovery and negotiation algorithms have not been investigated and there were no mechanisms to provide guarantee on execution completion. Scheduling of (batch) jobs on resources has been performed by metaschedulers, a Grid adaptation of successful schedulers for distributed systems like Condor [MJLM88]. This was sufficient for large batch and scientific jobs, but not for other new kinds of Grid applications with QoS requirements.

In particular, nowadays Grid Computing is becoming suitable for commercial applications, because of its ability to provide a common platform for service-oriented activities [MC04].

Such an economic scenario takes to an increase in the set of possible applications working on the Grid; and, consequently, the need to provide different levels of Quality of Service to the new classes of tasks, especially if "Grid resources" are sold for money, could be easily justified.

1.3 Introduction to Real-Time paradigm

Before dealing with time constraints on Grid, it is important to understand the basic concepts regarding jobs with deadline, namely the Real-Time paradigm.

From a general point of view, a system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed. This means, for instance, that given an event that activates the system, the system must respond within a maximum time from the happening of the activating event.

On the contrary, a non-real-time system is one for which there is no deadline, even if fast response or high performance is desired or preferred.

In particular, for the focus of this thesis, a system is considered real-time if it has a maximum length in term of operations to execute (in the worst case) and needs to be correctly completed within an absolute deadline from the moment of the activation. By contrast, a non-real-time systems is called Best Effort (BE), like a typical job of the first generation of Grid Computing.

In general, a Real-Time system can be defined as sets of jobs (i.e. sets of computational operations on a architecture). Any real-time job has a proper deadline to respect, coher-

ently with the global system deadline. On the other side, a Best Effort job, once activated, can complete in every moment.

1.3.1 Hard vs. Soft Real-Time

Real-Time systems are divided into two categories, Hard and Soft Real-Time.

The classical conception is that in a Hard Real-Time or immediate real-time system, the deadline is critical and the completion of an operation after its deadline is considered useless (and the miss could lead to the failure of the system).

In every-day life, humans are surrounded by hard real-time systems that, in case of failure (i.e. the miss of the deadline) could take to physically damaging or threatening human lives. An example can be a car engine control system where a delayed signal may cause engine failure or damage. Other examples of hard real-time embedded systems include medical systems such as heart pacemakers and industrial process controllers, or the airbag circuit of a car.

For a computing system point of view, Hard Real-Time systems are typically found interacting at a low level with physical hardware, in embedded systems. The reason for embedding Hard Real-Time systems could be likewise clear: because of the criticality, and, in some cases, the strictness of the deadline, the interaction between the hard real-time job and hardware must be direct, in order to avoid latencies (sometimes difficult to estimate or non-deterministic),

A soft real-time system on the other hand will tolerate lateness, and may respond with degraded service quality (e.g. dropping frames while displaying a video), in case of a deadline miss.

Soft real-time systems are typically used where there are some issues of concurrent access and the need to keep a number of connected systems up to date with respect to changing situations; for example, software that maintains and updates the flight plans for commercial airliners. The flight plans must be kept reasonably updated but can operate with a latency of seconds; in particular, in case of a deadline miss there are not critical nor dangerous situations, apart some sort of inefficiency in the service. Other examples are live audio-video systems: the violation of constraints results in degraded quality, but the system can continue to operate.

1.3.2 Real-Time vs. High Performance Computing

When dealing with Real-Time, the paradigm is often confused with high performance computing. For the aim of this thesis, and because we're focusing on Grid Computing, it

is important to clarify that fulfilling a deadline does not require necessarily high amount of resources, but, on the contrary, it is more important, as remarked, to have sufficient resources "in time".

For example, a high-end supercomputer executing a scientific simulation may offer impressive performance, but this does not mean that it is executing a real-time computation. Conversely, once the hardware and software for an anti-lock braking system has been designed to meet its required deadlines, no further performance are required (and so no other resources beyond the embedded system). Furthermore, if a network server is highly loaded with network traffic, its response time may be slower (decrease in quality of service) but will still succeed. Hence, such a network server would not be considered a real-time system: temporal failures (delays, time-outs, etc.) are typically small and compartmentalized (limited in effect) and do not cause catastrophic consequences. In true real-time system, such as the NASDAQ Index, a slow-down beyond limits would often be considered catastrophic in its application context (e.g. loss of money).

In some cases, high performance and Real-Time paradigm coexist in the same application. A typical example are chess-game programs. For instance, a chess program designed to play in a tournament with a clock will need to decide on a move before a certain deadline or loose the game, and is therefore a real-time computation, but a chess program that is allowed to run indefinitely before moving is not time-constrained. In both of these cases, however, high performance is desirable: the more work a tournament chess program can do in the allotted time, the better its moves will be, and the faster an unconstrained chess program runs, the sooner it will be able to move. This example also illustrates the essential difference between real-time computations and other computations: if the tournament chess program does not make a decision on its next move in its allotted time it loses the game (i.e., it fails as a real-time computation) while in the other scenarios, meeting the deadline is assumed not to be necessary.

Summing up, the most important requirement of a real-time system is predictability and not performance.

1.3.2.1 Real-Time predictability

In general, the predictability is the degree to which a correct prediction or forecast of a system's state can be made, either qualitatively or quantitatively.

In a computational context, this means that, given a real-time job it is fundamental to be able to provide a prediction on the completion of the job within a given deadline. Such prediction provides a probability of success (i.e. the completion within the deadline) on the resources the job is dispatched. In hard Real-Time systems, the probability of success has to be as closer as possible to 100%. In soft real-time systems, such probability provides

a measurement of the potential state of degrade of the system.

1.3.3 Real-Time and Grid Computing

Grid Computing can be characterized by a high degree of dynamism, faults and changes in users and resources. Such situation makes the Grid unsuitable for managing Hard Real-Time issues that, as noticed, are managed by proper embedded systems.

On the contrary, Grid Computing can be suitable for executing applications with both Soft Real-Time and high performance requirements, because of the huge amount of resources potentially offered by the Grid.

Supporting time constraints on Grid requires to raise significantly the QoS level provided by the Grid. In order to understand what does it mean in practice, it is important to analyze the current Grid state of the art from a QoS perspective.

1.4 Introduction to QoS on Grid

The new coming applications require more guarantees for an efficient execution in comparison with previous large batch scientific jobs.

Out of Grid, the QoS issue has been considered and solved in local systems like embedded ones [CLB06], and distributed systems like Internet [WZS00]. Unfortunately, the solutions achieved are good for distributed systems that are static (i.e. the set of users and resources does not change) and that belong to a single, well defined domain, where it is possible to assume a sort of total knowledge at every moment of the state of resources, users and jobs needs.

On the other hand, Grid is a dynamic and inter-domain environment where total knowledge is a wrong assumption: resources are heterogeneous, dynamic and belong to different owners while users change continuously as their job requirements.

For this, the management of QoS on the Grid is a complex problem that spans over all aspects of Grid from the architecture to the algorithms. Because of the growing demand of QoS, Grid had to adapt itself to support such needs; this took to an interesting evolutionary scenario driven by new QoS requirements that I analyze in the following.

1.5 Managing QoS issue on the Grid

Granting QoS on a system that is based on an architecture and algorithms that do not manage QoS is tricky and requires a deep extension of original features. In particular, every standard Grid functionality (i.e. discovery, negotiation and acquisition of resources, job execution) has to become QoS-aware. For this, it is important to answer the question: "*what does exactly mean to provide QoS on the Grid?*" and then understand how to enhance basic Grid features offered by current middlewares, so that QoS could be granted.

In this sense, a Grid supports QoS if at least the characteristics depicted in the following are added to the default functionalities.

1.5.1 QoS attributes, SLAs and metrics

First of all, it is important to define proper forms to express the offered QoS by the resource side and the requested QoS from the user one. From a practical point of view, this means to extend default middleware syntax and semantic with attributes that could be known and understandable by all entities involved.

QoS attributes can represent quantitative or qualitative characteristics. Quantitative characteristics refer to aspects that can be measured by some sort of sensor like CPU performance, network latency, memory or storage capacity. Some examples of attributes for CPU can be the Percentage of CPU time required and the CPU Frequency, while parameters for network QoS can be the Delay (the time a packet takes to travel from sender to receiver), the Delay Jitter (the time difference between packets traveling on the same route), Throughput (the number of packets sent in a given period of time), Packet-loss rate (the percentage of packets failed or lost). On the other hand, qualitative characteristics concern global aspects regarding comprehensive evaluations like service reliability or user satisfaction. Generally, the value of a qualitative characteristic should be expressed by a combination of quantitative ones, but the problem has not a simple solution and there is a sort of semantic gap between qualitative and quantitative characteristics; at present, in many proposals [RJAAW03] quantitative characteristics are preferred.

Attributes constitute the basis on which to build a QoS architecture in a real QoS context, but in general it is difficult to express requests and proposals in terms of attribute values and it is not necessary. In fact, a user is interested in obtaining guarantees that his job completes within a given deadline, he doesn't need to require to have the job running on a 2 GHz CPU for 5 seconds rather than on a 1 GHz one for 10 seconds. Moreover, it is not an easy task to matchmake on requests that are so strict.

For this, when dealing with QoS, requests and offers are made through *QoS classes* where at each class are associated performance conditions that permit the system to manage

adaptation. Examples of opposite QoS classes are the best effort, where there are no guarantees on performance (as the Grid in its original conception), or the guaranteed QoS where QoS requirements have to be maintained strictly.

Independently of the qualitative or quantitative nature, attributes must be contained in proper documents, in order to allow both parts to make offers (resource owners) and requests (user) through shared mechanisms, so that both parts could understand each other and come to an agreement on resource supply and application requests.

Currently, the standard for requirements and offers in Grid Computing are the Service Level Agreements (in the following, SLAs). Namely, SLAs "explicitly state the terms of the agreement between a resource user and a resource provider" [FK03], granting the possibility to define a shared contract (Binding SLA, BSLA) starting from the application requirements SLAs (Task SLA, TSLA) and resource SLAs (Resource SLA, RSLA).

Syntax of SLAs is not predefined nor standardized, but depends on choices from both sides. Semantic of a SLA depends on attributes contained, as well as the meaning of a SLA depends strictly on the significance of the single attributes it contains.

Although the idea of SLA was born together with the first Grid paradigm, SLAs significance and importance have increased with QoS management, so that SLAs are fundamental when dealing with QoS on Grid [Men04].

If SLAs permit to indicate which QoS level is needed or offered, the management of QoS requires suitable metrics to measure and evaluate the level of QoS, and so evaluating the fulfillment of QoS requirements in SLAs contracts. For example, typical metrics that allow application side to evaluate the level of QoS can be Average Throughput on job execution or the Queue Time (how much time the job waits before execution); on the other hand, a typical metric of resource side is the Average Resource Utilization.

1.5.2 QoS-based architectures

Besides a consistent and correct definition of syntax and semantic of QoS attributes and metrics for its evaluation, it is important to define proper architectures that grant to perform the operations of discovery, negotiation, acquisition of resources and jobs execution fulfilling QoS requirements. Such architectures constitute the component of a Grid system that manages the pool of resources, controls execution and interacts with requesting users and applications, taking into account QoS attributes. In the layered Grid architecture, it can work at Resource and Collective layers, just between the physical resources and applications. As we'll notice in the following sections, many approaches work over middleware, using its basic functionalities as a support for the management of QoS. Other approaches

are based on proper QoS middleware which directly implement QoS support.

Independently of the approach, current literature on QoS over Grid underlines that an architecture with support to QoS-based discovery, negotiation, and execution has to implement the following features:

- **Advanced Discovery**. The QoS-compliant architecture should support complex queries on QoS attributes with the aim to provide sets of suitable results for further selection operation (i.e. different BSLAs the application or user can choose among and then negotiate), like ranged queries on one or more QoS attributes.
- **Negotiation**. The system should provide protocols to allow a negotiation and a way to come to a contract.
- **Advanced Resource Reservation**. The system should support mechanisms for providing specific options for advance or on-demand reservation. Reservation is a strict requirement for granting the fulfillment of contract between parts.
- **Reservation Policy**. Resource owners must be allowed to select policies stating in which way and under which condition users can access resources.
- **Monitoring**. There must be protocols for monitoring the state of a contract. Such protocols calculate metrics and check if the contracts is fulfilled.
- **Adaptation**. In the case the monitoring protocols shown the lack of a term of the contract, the system should provide mechanisms to move current state to another that meet the established conditions. If this is not possible, the contract is violated.

- **Simplicity**. Adding QoS feature should correspond to simple design requiring minimal modifications, such that there are no changes in executing batch job without QoS demands.
- **Scalability**. Since Grid is a global-scale infrastructure, the QoS-enabled architecture should be scalable over the Grid.

Most QoS architectures on Grid implement only some of previous features, being specific for managing a particular operation (e.g. discovery or negotiation) while a few other proposals (frameworks) aim to manage any feature, sometimes directly but more often working together with other components.
In the following section, I present the most relevant proposals for managing QoS in Grid computing evolution, underlining which characteristics of the previous ones they implement.

1.6 QoS through the historical evolution of the Grid

1.6.1 Grid as large-scale resource sharing platform

In its first conception, the Grid has been uniquely defined as a platform for sharing resources. For this reason, any component concerning Grid, from the layered architecture definition [IFT01] to the first implementation of full working middleware [FK98] has been deployed with the unique aim to provide a real universal and common platform to share resources beyond the limitation of the single administrative domain, with no regard to the level of efficiency in matchmaking jobs and resources.

More specifically, from a QoS point of view, the focus of early Grid Computing has been the definition of a proper abstraction level (i.e. Virtual Organization) to share resources, while the management of such resources had a secondary importance.

As a consequence of this, discovery and negotiation algorithms have not been investigated and there were no mechanism to provide guarantee on execution completion. Scheduling of (batch) jobs on resources has been performed by metaschedulers, Grid adaptation of successful distributed system schedulers like Condor [MJLM88].

1.6.1.1 Condor-G metascheduler

Condor-G [FTL+02] has been one of the first Grid metaschedulers. It has been designed as a natural evolution of Condor scheduler [MJLM88] for working over Virtual Organizations and in a inter-domain way. Condor-G agent allows the Grid user to treat the Grid as an entirely local resource, granting to submit jobs and manage its execution (suspend/reprise). Moreover, it monitors job execution, providing a detailed log to user. The job monitoring is passive, and it is not implemented any adaptation or migration mechanism.

As a Grid metascheduler, Condor-G works over middleware (Fig. 1.2); it simply analyzes the state of virtual resources and then dispatches batch jobs on them, using underlying middleware from discovering (e.g. index services) till execution (e.g. Globus GRAM). In this way, Condor-G can use the Globus Toolkit [FK98] functionalities to manage virtual resources instead of physical ones as done in the previous version of Condor. Condor (similarly to other distributed schedulers) continues to be used but only in a given domain.

The main advantage of Condor-G is surely to be a first effective Grid metascheduler able to discover resources and run jobs on them, but it shows several weaknesses that is interesting to analyze and understand so that future Grid schedulers could be designed and implemented with improved quality and performance.
Condor-G shows both design and technical drawbacks. The main planning mistake is to

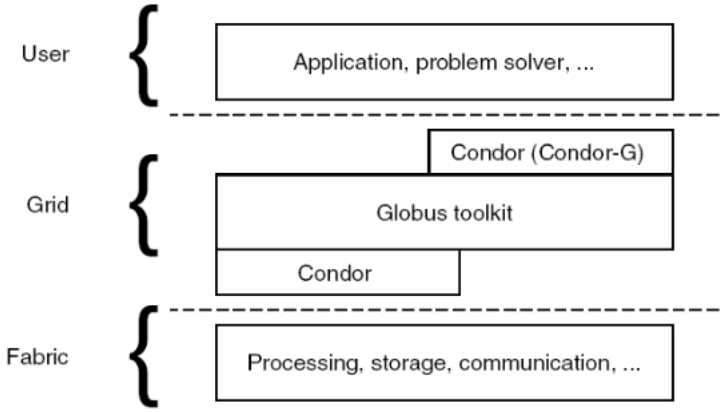

Figure 1.2: Condor-G in Grid layered architecture (taken from [FTL+02]).

have designed Condor-G following the same philosophy of Condor, that is thinking the Grid platform as a usual distributed system in a local domain. This formulation problem took Condor-G to have some features that are not suitable for efficient Grid Computing:

- *Centralized decisional process.* Condor-G scheduler is centralized: like Condor matchmaker, exists only a central unit that takes decisions on the whole set of resources and tasks; for this it does not scale well on big Grids with a huge set of resources, that can result difficult to manage in a centralized way.
- *Total knowledge.* The central decision unit works under the assumption of a total knowledge, that is a wrong assumption in a dynamic and heterogeneous Grid environment.
- *Single point of failure.* The single central scheduler constitutes even a single point of failure. If it goes down, other scheduling activities are not possible.
- *Strong assumptions at lower level.* Like Condor, Condor-G assumes that the execution of a job at Grid level corresponds at an automatic execution on physical resources.

Besides the design drawbacks, Condor-G lacks of any QoS support (i.e. the user cannot express any constraint on job execution), nor any early form of negotiation is supported.

Finally, Condor-G and any other first generation metascheduler [RBG00] did not obtain satisfying results because they were based on scheduling policies suitable for a distributed context in a single domain, but not sufficient to manage the real Grid complexity.

1.6.2 Grid as a Service Oriented Architecture

The first generation of Grid has been important because it provided the definition of an architecture for sharing resources universally, but was far from being an efficient system when implemented, because of the lack of standardization regarding resource representation and of algorithms to manage resource discovery and to matchmake jobs considering the submitting user constraints.

For this, research in Grid Computing focused on formal structures for defining resources and to build specifications like OGSA [IF06] for Grid-based applications. The goal of OGSA is the standardization of all services commonly offered by a Grid System (e.g. job management, resource, security, etc.) through the specification of a set of standard interfaces for such services. The definition of OGSA is still in progress.

OGSA took important modifications even to the definition of Grid middleware, whose aim became to implement the interfaces specified in OGSA. For this, Grid evolved toward a Service Oriented Architecture(SOA) [Ear05], where each element of the Grid is realized as a set of stateful Web Services (WS) (i.e. Grid Services), implementing interfaces of OGSA, following the WSRF specification [wsr].

Besides architecture, even different proposals for managing the functionalities provided by the Grid (discovery and negotiation of resources, job execution) have been developed, like the early resource specification languages. From a QoS perspective, nothing has been explicitly done in this phase, but some standards have been defined for supporting a further management of QoS like Service Level Agreements (SLAs) and Globus Architecture for Reservation and Allocation (GARA) [FKL+99].

The Service Level Agreements are documents that define a contract between a service requester (the Grid user) and a service provider (the resource, as a Grid Service). They allow both parts to define conditions that have to be maintained by both sides. In this second Grid evolution phase, SLAs have been adopted and defined to indicate user/task requests (TSLA), resource offers (RSLA), and to indicate a contract between an RSLA and a TSLA (BSLA); algorithms like SNAP [CFK+gh] to manage the negotiation phase through the use of SLAs have been defined.

The GARA architecture provides the first Grid resource management system (GRAM) with improved features like advanced reservation of resources and a first support to a trivial QoS management. Reservation becomes important after the *sign* of a SLA contract to avoid that resources involved in the contract could be allocated by other users. More specifically, GARA provides a uniform way to allow users and applications to manipulate both reservations and allocations.

1.6.2.1 GRUBER and DI-GRUBER resource brokers

GRUBER [DF] is an architecture and toolkit for resource usage SLA specification and enforcement in a Grid environment. It has been implemented both in pre-WS (first generation Grid) and in WS (second generation Grid) versions.

GRUBER works on a Grid made up of a set of resource provider sites and a set of submitting hosts. Each site contains a number of processors and a given amount of disk space. GRUBER considers a three level hierarchy of users, groups and VOs, so that each user belongs to one group only, and a group is a part of a VO only.

Resource usage SLAs (USLAs) are means that allow to govern the sharing of specific resources at different abstraction levels and among multiple users, belonging to different domains. Resource usage policies (in the following, RUP) space has three dimensions: resource providers (sites, VOs, groups of users), resource consumers (VOs, groups, users), and time.

GRUBER implements RUPs specification, enforcement, negotiation and verification mechanisms at each level and side, which have different final aim: Owners need convenient and flexible mechanisms for expressing policies on resources they share; VOs want to monitor SLAs; users and groups need to have guarantees on resource used to execute their jobs.

GRUBER supports batch jobs described by the processor time and the disk space required for execution plus the VO and the group of the submitting user.

GRUBER provides functionalities that allow it to work as a broker/metascheduler over the middleware, managing SLA-based resource usage policies and deciding how to matchmake resources with user jobs.

1.6.2.1.1 GRUBER architecture. Users submit jobs for execution at submitting hosts; GRUBER carries out the whole set of grid functionalities from discovery to acquisition, and runs jobs when enough resources are found.
GRUBER is composed by four principal components as shown in Fig. 1.3. The *GRUBER engine*, realized as a service, implements algorithms for detecting resources and maintains a view of resource utilization on the Grid; more specifically, it uses different logics to matchmake pending job with resource. The *GRUBER site monitoring component* provides data for the GRUBER engine; data are collected through a set of Globus and UNIX tools. *GRUBER site selector* is a tool that interacts with GRUBER engine to provide indication on which is the best site to execute a given job. Finally, the *GRUBER Queue manager* resides on submitting host and decides which waiting job to activate and when, monitoring the VOs policies.

DI-GRUBER (Distributed GRUBER) is the natural extension of GRUBER, able to work

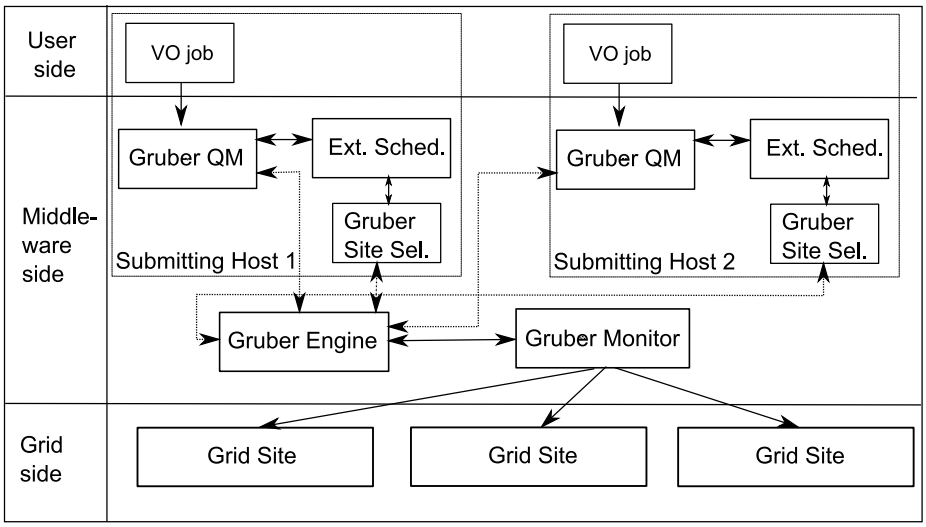

Figure 1.3: DI-GRUBER architecture.

on large Grids. In particular, DI-GRUBER extends GRUBER by introducing support for multiple scheduling decision points (in GRUBER there is a single one, the Grid engine) loosely synchronized via periodic information exchange. DI-GRUBER extends original GRUBER functionalities by adding distributed USLA management and distributed scheduling algorithms; moreover, DI-GRUBER has the ability to dynamically compute the correct number of decision points for the underlying Grid.

1.6.2.1.2 Analysis of GRUBER and DI-GRUBER. GRUBER acts like a metascheduler, discovering and matchmaking resources and jobs. GRUBER shows the same philosophy of earlier metaschedulers (e.g. Condor-G [MJLM88]) but working on open documents like USLAs. As a metascheduler, GRUBER selects when to start a submitted job and where, making choices instead of users and owners: in this sense, entities can indicate USLAs but are not directly involved in the scheduling process. For instance, a user cannot make a selection between different suitable USLAs nor decide when to execute a job; on the other hand owner does not receive feedback on their USLAs (e.g. how many have been selected or rejected) in such a way that it can adapt further USLAs and obtain a higher percentage of resource occupation. In addiction, GRUBER does not comprise any explicit QoS attribute or class.

For previous reasons, GRUBER and DI-GRUBER are good for managing batch jobs but not task with strict QoS requirements (e.g. computational steering, job with time constraints) because they do not implement mechanisms to guarantee job execution nor any QoS attributes.

However, GRUBER is important for QoS-based Grids because it is an interesting component to be implemented under a QoS framework, providing more efficient services for SLA discovering and negotiation than basic middleware or previous architectures like GARA [FKL+99].

1.6.2.2 AGMetS Framework

AGMetS [NLN+05] is a framework for QoS metascheduling that provides adaptations in case of fault during application execution. Every application managed by AGMetS has some QoS requirements and constraints on the minimum availability of the set of resources on which it executes. AGMets monitors running applications and in case of failure or low availability it migrates the applications on new resources.

Before the start of the application AGMetS considers the availability of the various resources. To obtain such information, AGMetS implements a "stated middleware" able to monitor resources and to be aware of their state. Moreover, it associates to every host in the Grid an *availability index vector*, specifying the failure probability of individual resources. Such vector is used to predict future resource availability of the site. This allows AGMetS to select the best set of resources where to migrate the application.

1.6.2.2.1 AGMetS architecture. As shown in Fig. 1.4, the main components of AGMetS are the User Interface (UI), the Information Aggregation Unit (IAU), the Availability Conscious Resource Management Unit (ACRMU), and the Job Submission Unit (JSU).

UI is an interface between the user and the metascheduler. IAU is responsible for gathering the resource information of various hosts on the Grid. ACRMU is the core component that matches the information collected by IAU with requirements of the job submitted through the UI and finds the most suitable host for that job. After a decision is taken, ACRMU passes the information to JSU that submits the job to selected host for execution. An Availability Program runs at each host pooling resources to the Grid. AP calculates periodically the availability index vector for its host and sends information to IAU.

When an application (i.e. a single or multiple job) arrives, upper middleware layers (over AGMetS) compute required availability from application's QoS parameters. Such index, along with application's other requirements, is given to AGMetS that chooses the host, and runs the execution on its selected resource. If, during execution, the availability index becomes lower than the value calculated before, AGMetS migrates the application to another host.

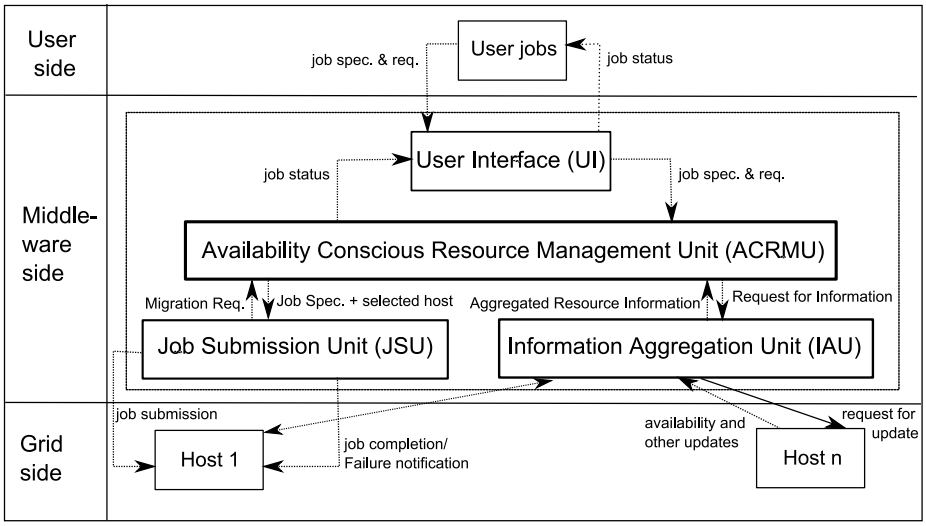

Figure 1.4: AGMetS architecture.

1.6.2.2.2 Analysis of AGMetS. AGMetS is an interesting approach because underlines the necessity to manage failures of resource. In first generation Grid, failure of resource results in the failure of the job. In batch context, at worst the job is restarted; in a QoS context, job failures could have different consequences, from delays (e.g. workflows executions) to loss of quality (e.g. Soft Real-Time jobs).

Differently from other approaches [AARW+02], it doesn't work over a general purpose middleware but it shows the need to have guarantees directly from the middleware (the stated middleware): with regards to the classification of QoS managers presented in the previous section, it is less *simple* because it requires modifications at middleware level.

AGMetS does not specify any QoS attributes or class; for this, it is not clear which kind of QoS application it could really support. It also does not support negotiation phase and policies because, as a metascheduler, it controls directly resources and jobs and make decision on matchmaking autonomously: in this case, like in GRUBER, both user and resource sides are totally passive.
Finally, it is a centralized approach (there is a single unit for the whole set of hosts and users), hard to scale on big Grids.

1.6.3 Towards a QoS-oriented Grid

The introduction of standards in Grid Computing, together with the need of QoS, had taken to the definition of the first effective architectures that provide QoS functionalities. Such frameworks use the standards (e.g. Grid Services) and basic functionalities (e.g.

security, authentication, information services and execution environments) offered by the middleware (e.g. Globus Toolkit 4), integrating them with advanced features for QoS management support.

In this section, I analyze both a framework for managing QoS attributes for single executions and an engine for manage workflows. In fact, the efficient execution of workflows over Grid constitutes a typical QoS issue.

1.6.3.1 G-QoSM framework

G-QoSM framework [AARW+02] is aimed at providing QoS for applications in a computational Grid. As a framework, it works over the middleware (Globus Toolkit) that uses for obtaining basic Grid functionalities. G-QoSM was initially designed for first generation Grid but evolved supporting both Grid Services [AAAvL+] and advanced features [AAHRW].

G-QoSM framework provides to the end user applications a service for discovery resources based on QoS parameters, with support both for QoS guarantees at three different abstraction levels (application, network and middleware) and for QoS management and adaptation on allocated resources.

G-QoSM manages QoS attributes for quantitative characteristics. More specifically, most relevant QoS attributes represent: 1) CPU and memory requirements (at middleware layer); 2) granted bandwidth and percentage of packet-loss (at network layer); 3) service reliability, availability and accessibility (at application layer).

G-QoSM architecture is composed of a well defined set of entities, whose interaction guarantees the QoS support.

1.6.3.1.1 G-QoSM architecture. G-QoSM main entity is the AQoS that works at the higher level, managing QoS at application layer. There is an AQoS entity for each application, that manages discovery through interaction with the two other modules NRM (Network Resource Manager) and RM (Resource Manager), as I can see in Fig. 1.5. AQoS manages negotiation and contract signature through the use of XML SLAs containing QoS attributes. Moreover, it monitors the state of the signed SLAs for its application, and manages violations of terms through adaptation techniques. NRM is a Bandwidth Broker and manages the QoS resources within a given domain. It is also responsible for inter-domain communication with the peer NRMs in other domains, and has the duty of coordinating SLAs across single domain boundaries. An RM manages single resources (as services) and has administrative and configuration control tasks; it acts like a combination of a Grid execution environment (e.g. GRAM) with a registry service which provides recordings of service capabilities and QoS provisions.

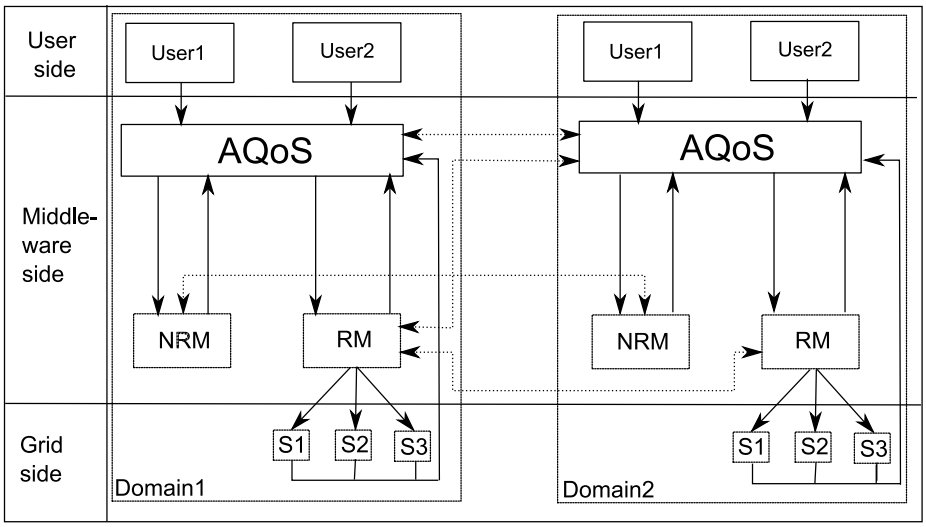

Figure 1.5: G-QoSM architecture.

G-QoSM supports three classes of QoS, namely guaranteed QoS, controlled load QoS and best effort QoS. In the first case, constraints related to QoS parameters of the client need to match exactly the service provision (i.e. in discovery phase only resources that provide at least the requirements of the client are considered, and the requirements have to be fulfilled for the whole application execution). In controlled load QoS, both in discovery phase and during execution, conditions are more relaxed and adaptation can be performed. Best effort QoS is the typical Grid QoS level, with no guarantees provided at any level.

1.6.3.1.2 Supported Grid functionalities. G-QoSM performs *advanced discovery* by interactions of AQoS, NRM and RM, and features like range queries (used in particular when controlled load QoS is required) are provided. *Negotiation, advanced reservation* and proper *policies* were originally supported by GARA in former versions of G-QoSM, while now are carried out through ad-hoc components [AaAL+04]; AQoS algorithms provide mechanisms for *monitoring and adaptation*.

Being a framework, G-QoSM works over middleware and does not require any modification of the middleware itself. As shown in Fig. 1.5, the G-QoSM framework has a *scalable* and distributed architecture, replicated in every administrative domain with proper components (AQoS, NRM, RM) for managing the QoS at a different abstraction levels. G-QoSM is able to manage highly demanding QoSs application like computational steering.

1.6.3.1.3 Analysis of GQoSM. G-QoSM implements most of QoS features on the Grid, but can be further enhanced. A first drawback consists in the assumption that the

user is able to indicate precisely its QoS requirements in terms of quantitative low-level resources only, without the possibility of managing qualitative characteristics (i.e. the user satisfaction). Another limitation is that it relies only on middleware for providing QoS at resource level. Finally, it does not provide support for managing deadlines on application execution.

1.6.3.2 QoS workflows: QGWEngine

The management of workflows over Grid is an interesting QoS demanding scenario since it requires the coordination of different services that execute the workflow. Moreover, workflows can be executed with QoS non-functional parameters, through proper architectures able to manage both workflows and to provide QoS support. The implementation of such architectures requires two stages: the negotiation and the scheduling execution. The first phase allows the user to reach an agreement on QoS parameters, while the second phase schedules the workflows over single services, matching previously negotiated requirements.

QGWEngine (QoS-aware Grid Workflow Engine) constitutes an architecture of QoS-aware grid workflow, which realizes previous steps. It supports five types of QoS attributes: *performance*, measured in terms of response time (request waiting time and request executing time); *cost*, namely the expense to invoke the workflow at a time; *reliability*, indicating the ability to execute the workflow under the given conditions; availability, the ratio of running time to total time of the workflow execution; *reputation*, the trust degree of a given service, depending on users' experience. QGWEngine supports workflows description written in GPEL that provides support for WSRF specification.

1.6.3.2.1 Architecture of QGWEngine. QGWEngine [WWD08] is divided in two parts: negotiation part and execution part. The first one implements a negotiation process based on QoS requirements between user and engine. The second one carries out management, execution and scheduling of a QoS aware workflow instance.

The Negotiation subsystem (Fig. 1.6) is composed of the proposal generator, the client side negotiation agent, the server side negotiation agent, and the proposal evaluation program. The proposal generator produces automatically negotiation proposals with QoS parameter values expected to satisfy workflow user requirements. Negotiation proposals are evaluated through the proposal evaluation program at user side that, given a certain proposal, rejects or accepts it taking into consideration the GPEL document with the workflow specification. When a proposal is selected, proper user and server side agents manage the negotiation process and come to an agreement.

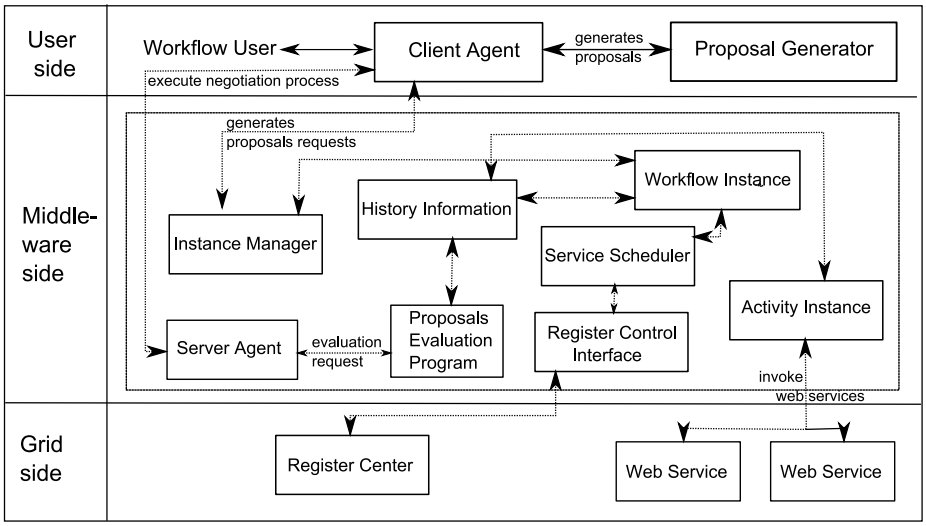

Figure 1.6: QGWEngine architecture.

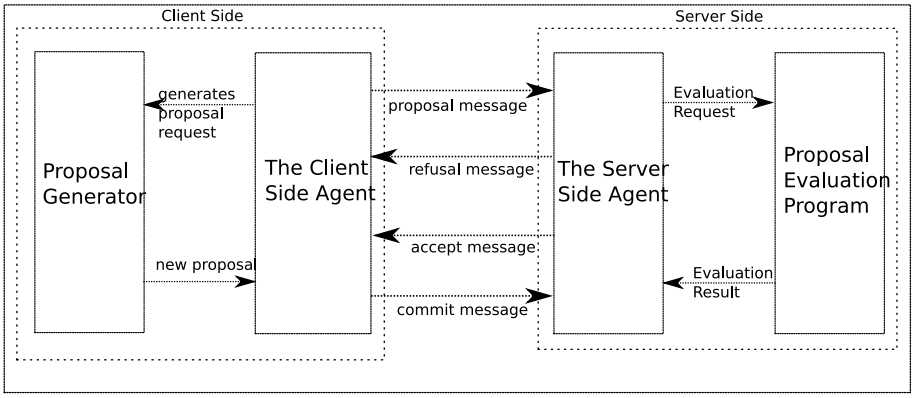

Figure 1.7: QGWEngine architecture: the execution phase.

The execution is managed by different parts (Fig. 1.7). Instance Manager realizes the scheduling, and the entire life-cycle of workflow instances, and a queue of requests. The Workflow Instance keeps track of workflow state and during the execution includes data and control logic. Every Workflow Instance is an independent thread that deals with different events, producing a set of activity instances that constitutes the smallest semantic unit and provides method for workflows execution.

The Service Scheduler manages the execution of component service of the workflow instance: it provides a series of component services that satisfy QoS requirements of the workflow. A brute force and a genetic algorithms are applied to decide the scheduling of the workflow instances.

1.6.3.2.2 Analysis of QGWEngine. QGWEngine main advantage resides in the ability to support QoS attributes for workflows, although only at application level, while the issue of QoS at lower levels is not considered. Moreover, it provides adaptation for scheduling workflows. Such feature is granted through a categorization of service kind that allow to swap execution from a service of a kind to another with same features.

The architecture is centralized; as a consequence, even in this case the scalability is an open issue. Finally, resource owners and users have a more active role with respect to other approaches because they have proper agents that work on their stead, managing proposal creation and negotiation phases.

1.6.4 Economic Grid for enabling a market on QoS supply and demand

The user demand of QoS and, on the other hand, the supply of QoS by the resource side formed the basis for the introduction of the market model for Grid Computing. In fact, as I noticed in previous frameworks like QGWEngine, the cost of services is a natural QoS requirement at application level.

Moreover, the introduction of economy in a Grid increases the participation of resource owners that are, in this way, more stimulated to offer resources for a profit. In a market model, owners can offer their resources at different QoS classes, requesting different prices. On the other hand, users should spend to acquire and access resources with required QoS level; under this point of view, their main goal is to obtain what they need at *lowest price* and *in time* for their execution requirements.

Such scenario introduces elements of both *cooperation* and *competition* in Grid Computing:

active cooperation and proficient interaction between user and owner sides is a fundamental requirement to build up a functional market based on offer and supply and active negotiation phases. Moreover, competition is a necessary consequence of a market model: in fact, user entities compete each other to acquire better offers while, on the other hand, owner entities compete to support better user side's requests.

The introduction of economy in a Grid has different consequences in a QoS context:

- **New attributes management**. QoS framework need further extensions in attribute syntax and semantic to support economic model (e.g. SLAs with cost). Moreover, proper algorithms, frameworks, services for performing cost-based discovery and negotiation have to be implemented.
- **Utility function definition**. Both user and owner sides have to implement utility function algorithms for binding resources and QoS offers to prices (in owner's resource SLAs), and to a maximum cost affordable for the QoS requirements (in user's task SLAs).
- **Definition of payment systems**. Economic Grid requires payments between parts. In this sense QoS framework need to implement systems for managing secure transactions. Such systems can be offered by low level middleware services (e.g. Grid security services like authentication and authorization) or at framework level.

The first issue to address in realizing an Economic Grid is to understand how to introduce economy in a well-defined architecture like Grid, originally designed with no economic support. To this aim, a first step requires to clarify what does it mean to implement economic models in Grid. Although the price/cost concept could be inserted simply as a QoS attribute on Grid resource brokers (cfr. 1.6.2.1) or QoS frameworks (see 1.6.3.1), a full implementation of economic models is a more complex issue. In particular, a market scenario cannot be managed by a single decisional point, but it must be implemented through interactions of entities which represent the different stakeholders in the market (owners and users at first, but also any other entity that can have earn in supporting their activities).

Moreover, while in classical Computational Grids the choice of resources for job execution is done once and before the start of execution, an Economic Grid must support the possibility to change dynamically the choices made, depending on application need and current supply of the market.

Finally, it is not an easy task (nor a common point of view exists) to understand at which abstraction levels in the Grid layered architecture [FK03] to implement and manage market models; more specifically, there is the need to answer the questions "how invasive

is economy in Grid architecture? Is it possible to maintain a simple (cfr. 1.5.2) approach?" In the following, I try to find out answers by analyzing two different proposals.

1.6.4.1 GRACE framework

GRACE (GRid Architecture for Computational Economy) infrastructure is a middleware component that works with basic middleware like Globus, providing services that help centralized Grid Resource Brokers (e.g. Nimrod/G [RBG00]) or independent owner and user agents to support dynamic computational economy.

In this sense, GRACE constitutes a sort of module that expands middleware functionalities with the support of economics, through a proper set of APIs and trading protocols that allow users to trade resources potentially in every moment, even during their jobs execution.

1.6.4.1.1 GRACE infrastructure GRACE infrastructure is based on the following components: a Trade Manager (TM), that constitutes the user side client or the component of a resource broker used to perform request (an then discovering) of resources; a Trade Server (TS), that works on resource owners behalf, using algorithms for calculating prices for offered resources, negotiating with users and selling access to managed resources; Trade Managers and Trade Servers interactions for trading resources are granted by proper trading protocols.

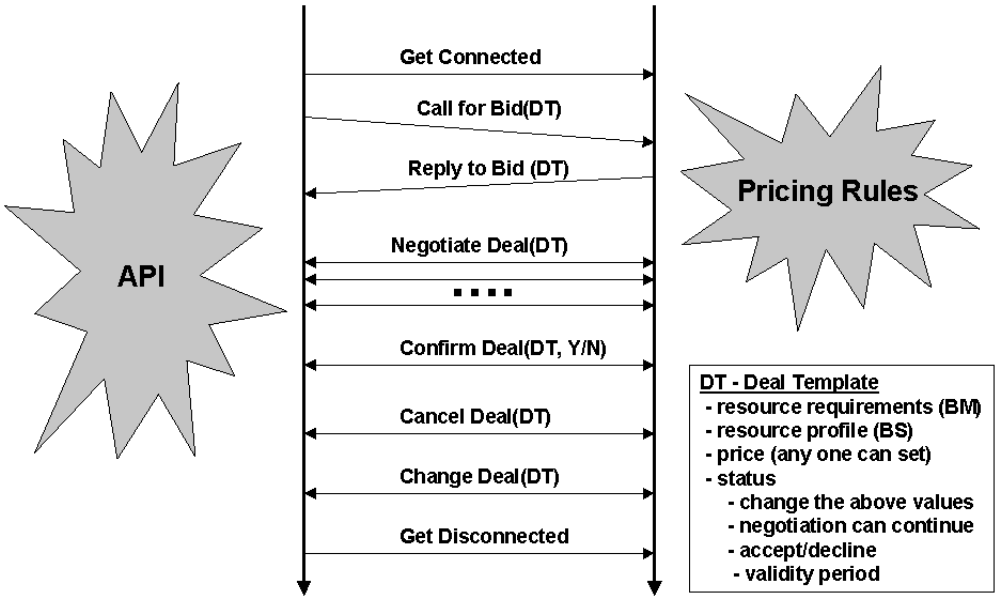

Figure 1.8: GRACE trading protocol (taken from [gra]).

The TM works under the direction of a Grid Resource Broker that, taking into account user requirements, searches for resources interacting with a set of TSs for retrieving better offers. Every TS has the aim to maximize its owner profit and resource utilization, and it's behavior is dependent on current Grid demand and supply ratio; among different TSs a "competition" exists for proposing better offers.

Trading Protocols shown in Fig. 1.8 grant interactions between TMs and TSs, working as follows. A TM contacts a TS with a quote/bid request, specifying a proper document called Deal Template (DT) which contains requirements (CPU, storage, RAM, prices) described in a proper language similar to *ClassAds* of Condor [MJLM88], and sends it to TS. TS proposes a matching DT and so a negotiation phase between TM and TS is performed in a undefined number of interactions and ends with a possible deal confirmation. After the confirmation, TM and TS interactions could continue to change or deleted the previously agreed deal.

1.6.4.1.2 Analysis of GRACE GRACE is the first architecture that introduces and manages a cost model in Grid, achieving a first step towards the definition of the Economic Grid.

In such approach, prices, costs, demands and supplies are managed at a over-middleware level, making the economy introduction simple and not invasive in the lower Grid levels.

However, economy management is still trivial, there is no implementation of advanced economy models, and the only stakeholders in economy are final users and the owners of resources; in this sense, GRACE introduces economic attributes (like cost and price) but does not support yet a comprehensive and complex market based economic world.

From a QoS point of view, QoS is not directly managed, although the approach could support quantitative QoS attributes as an extension of Deal Template Specification Language; qualitative QoS attributes or requests are not supported. Moreover, advanced discovery and reservation are not explicitly investigated, but could be both supported: advanced discovery can be added by proper overlying framework like G-QoSM [AAAvL$^+$03], while advanced reservation can be granted if an underlying middleware, used by TS to discover the state of controlled resources, supports it (like, for instance, Globus Toolkit 4 [Fos06].

1.7 Global analysis of the state of art and open issues

The previous analysis of most prominent and different approaches in QoS over Grid underlined some interesting aspects, especially if the current state of art is compared with the historical evolution of the Grid. In particular, QoS changed all aspects of Grid, from the

role of owners and users till the way to implement the Grid basic functionalities (discovery, negotiation and execution).

In its original conception, Grid Computing involves three main entities: the resource owner, the grid user and the metascheduler. In this context, the metascheduler acts as a sort of broker between user's demand and owner's proposals assigning pending tasks to available resources, following its proper policies. In this way, both the usage level of resources and the number of jobs completed depend only on choices made by the metascheduler that is the only active entity, while users and owners are passive and have a marginal role.

On the other side, when dealing with QoS, both owners and users acquire a more important role. The owner defines actively not only the access policies but even the usage policies (i.e. in which way a resource can be used and under which execution constraints is provided). Moreover, in this context the owner's aim is to have as many resources occupied as possible.

On the other hand, the user has to execute jobs with constraints that it has to manage. This means that the user must be able to discover and find proper resources in the moment in which it need them. For this, user has to make active choices between possible alternatives with the aim to satisfy as constraints as possible, granting the proper completion of its job. Such new trend leads to three fundamental consequences:

- **Two-sides**. Grid Computing scenario depends more and more on active interactions between owners and users, while centralized or distributed metaschedulers are loosing their former important role. As a proof of this new trend, metaschedulers have evolved into frameworks (e.g. [AARW+02]); some frameworks still decide in stead of users in some cases under the assumption that the user is able to indicate precisely its needs. In general, there is the trend to emphasize this independent duality through the introduction of autonomic agents [WWD08], and to support the interactions of both sides through proper services that attract other different stakeholders to Grid (cfr. GridEcon Project [AR06]).

- **Economy**. The possibility to offer the same resources at different conditions and the request of many constraints by user side enable a real economic system on the Grid where the owner sells resources with different levels of QoS at different prices; on the other hand, the Grid user asks for resources and spends depending on services it receives. In the "middle", many economic models and entities could work.

  The new coming Economic Grid [BAG01] is a clear witness of this new approach to the Grid. In an Economic Grid owners sell services (QoS) on resources with the aim to obtain the maximum gain, while user's aim is to obtain the required resources at minimum cost possible. As a result, some QoS frameworks like QGWEngine start to implement cost as a QoS attribute at application level and explicit support to economy, while previous ones (like G-QoSM) did not take into account any economic trend.

- **Cooperation and competition**. The context presented in the previous point could take the Grid to a deep change in its evolution, moving it from a collaborative and disinterested context to a new competitive and gain-oriented one. Autonomous owners and users cooperate each other to obtain the better result for each side. However, every single owner competes with others to provide to users a better offer with the aim to maximize the resource usage (i.e. profit). On the other hand, even users vie with one another to get the best resource offer.

  The entry of support for autonomous agents [JC02] in Grid Computing underlines this new coming scenario: formally, in computer science, an intelligent agent (IA) is a software agent that assists users and will act on their behalf, in performing non-repetitive computer-related tasks, using a form of artificial intelligence it possesses. Agents nature fits perfectly the new *active* Grid context: agents represent users and owners and pursue their respective aims, interacting with peers and other side agents, and making choices.

  Another consequence of such all-round competition regards even the forming of Virtual Organizations. While in first generation Grid Computing VOs were organized to support long term experiments or projects, new VOs can be built only for the last of a contract between some owners and users and are destroyed when the contract expires. In this sense, VOs could become more dynamic and transient.

In the depicted scenario, the way of defining QoS attributes syntax and semantic is a critical issue and negotiation assumes a key-role, while in first generation Grid this phase was not considered at all.

With respect to the classification made in section 1.5.2, the analysis of Grid evolution under a QoS perspective takes necessarily to *improve* the set of requirements for an "efficient" QoS management:

- **Universality**. SLAs constitute a set of documents that together with the use of standard and common languages like XML provide a format for publishing offers and requirements. Nevertheless, SLAs do not provide any constraints nor indication on their contents, so the choice on the way to representing data and their semantics is left to the writer of the SLA. In this way, there is no standardization in resource information nor in application constraints, so a SLA is not universally understandable but only by a proper underlying Resource Management System. For this, when providing support for QoS, there is the need to use syntax and semantics well understandable by every owner or user entities involved in providing or requiring QoS.
- **Entities independence**. As noticed, the role of Owner and User changes, granting them an increasingly importance in QoS based resource offer and selection. Such

reported growing trend is justified by the consideration that, in a Grid environment where diversification of requests comes together with different ways to offer the same resources, there is the natural impossibility for a super-part entity (like early metaschedulers) to make choices efficiently. In this context, it is a natural consequence that more QoS level is offered in a Grid, higher will be the independence of owner and user, and a single super-part decisional point must be avoided.

- **Qualitative approach and semantic gap**. QoS parameters can be quantitative or qualitative. First type parameters are typical of lower abstraction levels in Grid architecture, while second ones regard higher application levels and users. As shown, qualitative parameters are preferred at final user side for two reasons: first, users do not have precise requirements (e.g. 10.000 CPU cycles) but more general ones (e.g. application completion before a time T); secondly, a qualitative request increases the number of possible matching resource offers, because there is no need to match exactly a set of provided quantitative attributes, but different sets of quantitative attributes can fulfill the qualitative request. In this context, a very complex open problem is how to efficiently and significantly translate qualitative user requirements into quantitative low level resource characteristics. Namely, this means how to produce significant BSLAs from a set of RSLAs (also called low level SLAs) and TSLAs (also known as high level SLAs). This situation indicates a deep semantic gap between Application level and Physical level in the Grid abstraction model.

- **Cross-Layering**. The semantic gap problem and the need of universality underline that QoS in Grid is surely a cross-layer problem [LCL07a] that comprehends all levels of the Grid hourglass model [FK03]. In fact, it is not possible to obtain general QoS at a certain level (e.g. application) if every parameter at that level cannot be automatically translated in a set of parameters of the underlying level and so on until the physical one.

- **Economic need**. As shown before, managing different QoS requires a proficient participation of resource owners, as independent entities, both in quantity and in quality. Quantity means that many owners must be stimulated to share resources, and quality means that, on a possible huge set of resources offered, a satisfying set of offers has to be made to fulfill the heterogeneous user demands. In this sense, owners are stimulated to actively participate if a sort of profit can be granted to them. On the other side, users with QoS requirements must be limited in their requests (an egoistic approach where more resources than needed must be taken into account and avoided in a context of independent agents); in this situation, binding resource to a price, and conversely user to an expense could implicitly manage such behaviors. Under this point of view, the evolution towards an economic model is a natural need of Grid, and it is difficult at present to suppose a future Grid oriented to QoS without an economic support.

- **Invasiveness**. In the analysis of different approaches, it is possible to notice that together with the growing demand of QoS and the strictness of new coming QoS requirements (e.g. from early best-effort to current workflow synchronization), the range of abstraction levels involved in QoS management moves towards lower ones; this means that to provide support for stricter QoS constraints, some guarantees have to be provided already at middleware and local resource levels. As a consequence of this, the simplicity can not be granted (e.g. it is not possible to manage QoS only at a framework, over-middleware level, as remarked); on the contrary, higher is the level of QoS required, more guarantees have to be provided by owners directly, meaning that QoS must be addressed, in many cases, from the resource level itself.

	Pre-QoS phase			QoS phase		Econ. phase
	Condor-G	(DI-)GRUBER	AGMetS	G-QoSM	QGWEngine	GRACE
Advanced Discovery	no	no	yes	yes	yes	yes
Negotiation Support	no	yes	no	yes	yes	yes
Advanced Reserv.	no	no	yes	yes	no	no
Reservation Policy	no	no	yes	yes	no	no
Monitoring	yes	yes	yes	yes	yes	no
Adaptation	no	no	yes	yes	no	no
Scalability	no	DI-GRUBER	no	yes	no	yes
Open Standards	no	yes	no	yes	yes	no
Entity Independance	no	no	no	no	in part	no
Qualitative support	no	no	no	no	yes	no
Cross-Layering	no	in part	yes	no	no	no
Economy	no	no	no	no	no	basic
Simple vs. Invasive	simple	simple	rather inv.	simple	simple	simple

Figure 1.9: Comparison of the analyzed approaches with respect to a set of features.

As a final remark of this analysis, I point out in Fig. 1.9 which characteristics belonging to the extended set of requirements needed for an efficient QoS management, the various approaches analyzed in Sect. 1.6 support.

1.8 Analysis of Real QoS applications on Grid

The previously analyzed approaches offer many services and features for supporting different QoS levels and constraints. Such approaches are effective when providing support for real applications with QoS demands. For this, the aim of this section is to present two kinds of applications with different QoS requirements that could exploit the benefits of a Grid with QoS support.

In particular, for any kind of application, a brief introduction and a justification on benefits in case of migration on Grid is provided. Afterwards, an analysis of QoS features (among those presented in Fig. 1.9) needed by each application is provided.

The final aim of this section is to answer to the questions "is there an approach that is able to fully support such real applications?" Otherwise "how can be managed efficiently the QoS for a new coming Grid application?"

1.8.1 Massive Multiplayer Online Role Playing Games (MMORPGs)

A MMORPG [FBS07] is a particular kind of a Massively Multiplayer Online Games (MMOGs) that corresponds to a massively online virtual transposition of Role Playing Game where, in a large world, several thousands of proper characters (that are the projection of the real player in a fantasy world) perform activities with the aim of acquiring *experience points* in order to evolve and increase its abilities in the game.

Actions of a single player are performed on single user machine but the whole world, the interactions between players, the timeline and data on players' status are managed by a proper set of game servers belonging to the private organization to which MMORPG belongs.

In any moment, a high and varying number of concurrent players need to be managed, and the end-user perceived performance depends directly on the way the MMORPG server architecture is able to scale, with respect to the number of current players.

1.8.1.1 Open issues in MMORPG

At present, the choice on resources measuring and allocation for game servering is made once, statically, before the game becomes online. In this case, the number of maximum supportable subscriptions is fixed; if the need of resources is underestimated, the management of further subscriptions (and potential active players) implies a reduction of performance or the impossibility to accept new subscriptions.

On the other hand, if the number of subscriptions is always considerably lower than the supported one, there is a waste of resource usage. In general, in the lifetime of a MMORPG the number of active subscriptions (see Fig. 1.10) varies considerably. This mainly depends on the fact that subscriptions in MMORPG are sold monthly and a player can decide to play only for some months, suspend the account and then come back successively to play again, renewing the subscription. On the contrary, in all other genres of MMOGs, the access to the game is sold once so the number of potentially active players never lowers over time, making resource measuring not an interesting issue.

Moreover, in MMORPG subscriptions trend can also depend on other aspects (e.g. success of the game, releases of game expansions, MMORPG market trend), that I do not investigate here.

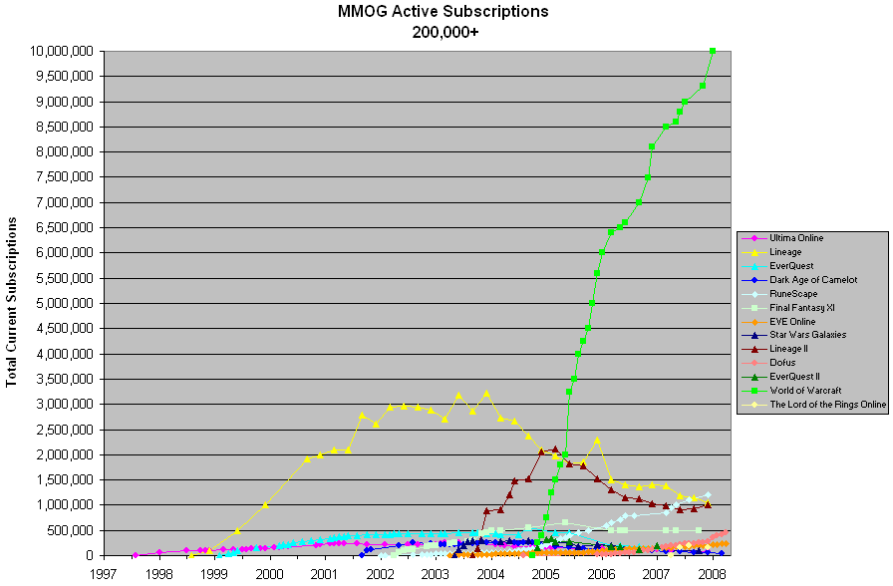

Figure 1.10: Trend of subscriptions in most relevant MMORPG in the last decade (source: http://www.mmogchart.com/).

1.8.1.2 MMORPGs on Grid: motivations

In last years, an emerging point of view is that Grids could be used as a global external provider for acquiring resources for massively online gaming [DHB+06], realizing effectively a dynamic resource provisioning. In this context, a fundamental assumption is that some kind of QoS could be granted at Grid side, especially for MMORPG (e.g. constraints on minimal bandwidth and computational power for managing N players or a given instance of the map), that constitutes an harder issue from a game server point of view [MSG05].

Moreover, another conviction is that dynamic resource provisioning could become a killer application for Grid[LCL07b], providing the basis for the birth of an open market of resource providers for MMORPGs, where MMORPG owners pay for renting resources, and resource owners earn in providing resources with the required QoS level.

On the frameworks side, some general QoS management proposals have been recently used for supporting MMOG resource provisioning on Grid, like OptimalGrid[LK03], while others have been designed explicitly for managing players and game sessions on Grid, like Real-Time Framework(RTF)[GFGMI+08].

In this example scenario, I consider the RTF as an instrument for managing MMORPG game sessions and players over resources obtained from a ordinary Grid. I am not interested in exploiting RTF functionalities, but only in showing which QoS features it expects to obtain from the underlying Grid to work efficiently.

1.8.1.3 QoS requirements of RTF

The Real-Time Framework manages the MMORPG dynamic resource provisioning issue. More specifically, it manages the discovery, the negotiation and acquisition of Grid resources for dynamically scaling game sessions.

The Discovery phase is based on *quantitative QoS attributes*, regarding guarantees on CPU and network bandwidth that are required to independent resource providers (namely, the Hosters). Because RTF aims to be a reference platform for whichever MMOG, standards used for indicating such attributes are *open* and *universal*. Therefore, a system like RTF needs the support of a mechanism of *Advanced Discovery* for lower Grid layers, different from basic discovery services of general Grid middleware.

Moreover, the autonomous nature of resources requires an *active negotiation* between RTF and Hosters, plus the ability to support interactions between such *independent entities*. Another characteristic of RTF is that resources are retrieved when needed, so there is no need for *advanced reservation* mechanisms from lower Grid layers.

Once a resource is acquired and accessed, since in MMORPG the request of resources is, in every moment, in linear dependence with the number of active players[YC06], there is the need of *monitoring* and *adaptation* services.

The depicted system works well in an *economic context*, where the supply of resources from Hosters could be boosted by an earning expectation. This possibility would stimulate such independent entities in fulfilling RTF requests, competing one another for signing the best contracts with the RTF Scheduler.

Finally, a system like RTF requires a QoS support that is necessarily both *cross-layer* and *invasive* for the Grid middleware. In fact, RTF requests regard a set of combined

resources (e.g. CPU, RAM, bandwidth); the QoS level of the set of resources involved depends strictly on the QoS provided by the single resource composing the set. Moreover, such QoS has to be monitored on every single resource. All this implies that QoS must be supported both at application level (RTF) and at middleware level, where QoS attributes and support must be provided on single resources, making mandatory the management of QoS on different abstraction levels from Application to Resource in the Grid layered architecture[FK03].

1.8.2 Urgent Computing

Even though scientific jobs for Grid Computing are generally *best effort* without any need of completion within a given deadline, is it also true that many operational applications, exploiting a scientific infrastructure, are tied up with external events (like natural ones) that determine the application contingent behavior.

As examples, earthquake predictions[ea02a] and climate modeling could require a peak of computation when critical situations occur, that need to be managed within a given deadline. Such situations can be tackled by a new and evolving computational field called Urgent Computing[Don08].

Urgent Computing aim is actually granted by the improved fidelity and utility of high-performance computing to decision making. It refers to the concept of providing prioritized and immediate access to ordinary supercomputers and Grids for emergency computation. A typical Urgent Computing application cannot waste time waiting in the job queues and needs access to compute resources as soon as possible.

Urgent Computing can particularly benefit from Grid platforms; in particular, its high number of resources available could grant both heterogeneity of resources and redundancy, so that failures of resources running a critical job could be solved by migrating the job to another available resource.

For the aim of this paper, I reduce the complexity of Urgent computations considering an urgent computing job as a job that is activated with a low level of predictability and needs to *complete within a given deadline*.

Like MMORPG, Urgent Computing implicitly requires QoS-based services from lower level Grid layers. To better understand and justify such requirements, we'll briefly analyze SPRUCE, the first Urgent Computing framework.

1.8.2.1 SPRUCE

The Special PRiority and Urgent Computing Environment (SPRUCE)[BNTB06] is a token-based framework that allow urgent computation to interface with Grid and access some resources with high priorities.

When dealing with urgent computations it is fundamental to consider four basic points: 1) a precise session in which the urgent computation can be performed must be defined; 2) each computation could be easily frozen and transferred on other resources for adaptation purposes; 3) each user is associated with a set of resources on which to run the job, and he needs to be free of submitting the job on any of such permissible resources; 4) the framework that manages urgent computation must support computing policies that can be implemented on Grid resource providers.

SPRUCE fulfills the above requirements through the idea of token. A SPRUCE token is a unique ticket that corresponds to an association with a given set of resources. It is created with a lifetime that specifies the duration of the urgent computation session. When a token is activated within the lifetime period, the access to the associated set of resources is granted for the execution of the urgent computation.

However, the main focus of SPRUCE is not to execute the urgent computation after a token is activated, but to provide a platform that assists the user in choosing the best set of resources (i.e. which resources must be activated) so that the urgent job could be completed within its deadline.

SPRUCE supports all this by following a proper procedure. Given the set of accessible resources identified by the token, a SPRUCE component called Advisor calculates a probabilistic upper bound on the total turnaround time for the urgent computation on a certain resource. The idea is to express the probability that the total time from staging to execution on the chosen resource is less than or equal to the turnaround time.

Given a set of resources, the system evaluates all possible configurations (i.e. the possible executions on different resources and with different urgency levels) so that the user can choose the best one when the urgency occurs. This choice is then made by the user when urgent computation needs to execute, and it is not supported by the SPRUCE framework.

1.8.2.2 QoS-Grid and SPRUCE

Considering the SPRUCE behavior presented, it is clear that SPRUCE works well on resources that have particular characteristics (e.g. allow pre-emption, are monitorable, support urgency computation). Because middleware implementations of resources (e.g. as Grid Services) lack in providing such advanced features, in this section we'll analyze which characteristics reported in Fig. 1.9 are required for supporting SPRUCE by an ad-hoc QoS

framework.

First of all, SPRUCE does not need any *advanced discovery* nor *negotiation support* as the assignment of user with urgent job to resources is made statically, defined by the set of tokens belonging to the user and does not change.

Moreover, the token corresponds to a privileged access and reservation on resources within the time session, during which resources can be accessed under different urgency levels, so *advanced reservations* systems and *policies* are needed to grant an effective access and usage of the token associated resources when an urgent computation has to be executed.

As remarked, a token comprises a set of resources and a user with urgent computations can have different tickets for different sets of resources where potentially run the urgent computation once activated. Before the urgent computation is activated, all sets of resources are monitored in order to check the availability of the single resources and, in case of problems, a resource can be considered no more suitable for further urgent computations. For this, *monitoring* services are needed.

Monitoring activities are performed even during the execution of a computation so that problems on resources where the job is running can be solved by freezing the computation and migrating it to other resources. For this, *adaptation* mechanisms are required by SPRUCE.

SPRUCE also needs a *scalable* support on the Grid because resources contained in a token can be potentially spread over the whole Grid, and there is no a central point for managing the whole set of urgent computing users.

As SPRUCE support for urgent computation can rely on any kind of possible Grid resources, the use of *open standards* by lower levels QoS frameworks is a strict requirement so that SPRUCE work effectively on a real Grid.

User and owner sides are not considered as *independent entities* as there is not any interaction or negotiation activity among them.

Job requirements are explicitly quantitative (e.g. CPU, RAM, disk space), so a *qualitative support* is unnecessary. Moreover, job execution time and deadline are calculated as sum of basic Grid operations (input/output file staging, execution) where every operation regards a well-defined subset of basic resources (e.g. staging regards network bandwidth and disk space, while execution regards CPU). In the same way, the potential QoS requests in SPRUCE involve quantitative QoS requirements on basic Grid resources, making the QoS issues not a *cross-layer* problem (i.e. QoS is considered only at Resource level).

SPRUCE assumes that policies for accessing resources during an urgent job execution are defined at the resource owner level, provided directly by the owner itself. For this, in my definition, SPRUCE approach is *simple* because it does not need any improvement of Grid

middleware functionalities.

Finally, SPRUCE could be supported better on an *economic* Grid, because, also in this case, the supply of resources for supporting different urgency levels (and then different QoS constraints) could be stimulated by an earning expectation by owner side. Unlike MMORPG, here economy is not a stimulus for competition among owners (resources are passive in this case), but only an incentive for owners to provide support for urgent computations.

1.8.3 Real applications and current QoS-frameworks

The analysis of both RTF and SPRUCE shows that real Grid applications can have different requests on QoS support and services. Fig. 1.11 summarizes the QoS services required by both applications.

	Real Time Framework	**SPRUCE**
Advanced Discovery	**X**	
Negotiation Support	**X**	
Advanced Reserv.		**X**
Reservation Policy		**X**
Monitoring	**X**	**X**
Adaptation	**X**	**X**
Scalability		**X**
Open Standards	**X**	**X**
Entity Independance	**X**	
Qualitative support		
Cross-Layering	**X**	
Economy	**X**	**X**
Simple vs. Invasive	**invasive**	**simple**

Figure 1.11: QoS services required by RTF (MMORPG) and SPRUCE (Urgent Computing).

At a first observation, it is clear that RTF and SPRUCE have different QoS support

requirements. In spite of this, none of the analyzed framework can fulfill by itself all requirements of either applications, as I can see comparing Fig. 1.9 and 1.11.

In this sense, only a combination of functionalities from different frameworks can be sufficient. For instance, GQoSM supports many of RTF features but lacks in providing economy management so that a free market of resources, that constitutes one of the aim of RTF, could result difficult to be realized. On the other hand, GRACE approach provides support to an economy market but does not provide any QoS support. Similar considerations can be easily done on SPRUCE.

This fact has some consequences: first, current approaches are not sufficient to manage real applications with QoS demand on Grid. Moreover, the analysis of RTF and SPRUCE requirements underline that possible Grid applications can have very different demand in terms of QoS. For example, RTF performance is strongly based on efficient advanced discovery, negotiation support and entities independence in order to manage a free market of *on demand* resources. On the contrary, SPRUCE has no need of such features, but requires higher efficiency reservation services for granting a persistent control on resources and an immediate access when the urgent computation starts.

Finally, even if two applications can require the same QoS services (Advanced Discovery, Negotiation Support, etc.) the single features can have different meanings and could require unequal support. For example, both SPRUCE and RTF require adaptation support but with different aims; SPRUCE adaptation regards the migration from a set of resources to another one if some failures on resources is recognized before the urgent computation starts. In RTF the adaptation regards the migration of game sessions at run time, when the sessions result too full of player and need to scale dynamically on new resources.

1.8.3.1 Explicit QoS management on Grid

In general, previous considerations underline that it is impossible to assume (and then define) a single QoS framework able to manage the whole set of possible new coming QoS-aware Grid applications.

On the other hand, the demand of QoS from applications that aim to run on Grid has become considerable, making QoS a core aspect of Grid Computing and, in general, of Service Oriented Architectures. Besides, the analysis of the QoS proposals shows a lack in a standard approach.

At this moment, it appears clear that QoS has to be considered explicitly in the Grid paradigm, using a common and standard approach. From the analysis made previously, it is my opinion that a proper abstraction layer for addressing QoS issues and executing QoS-frameworks for different classes of applications needs to be formalized.

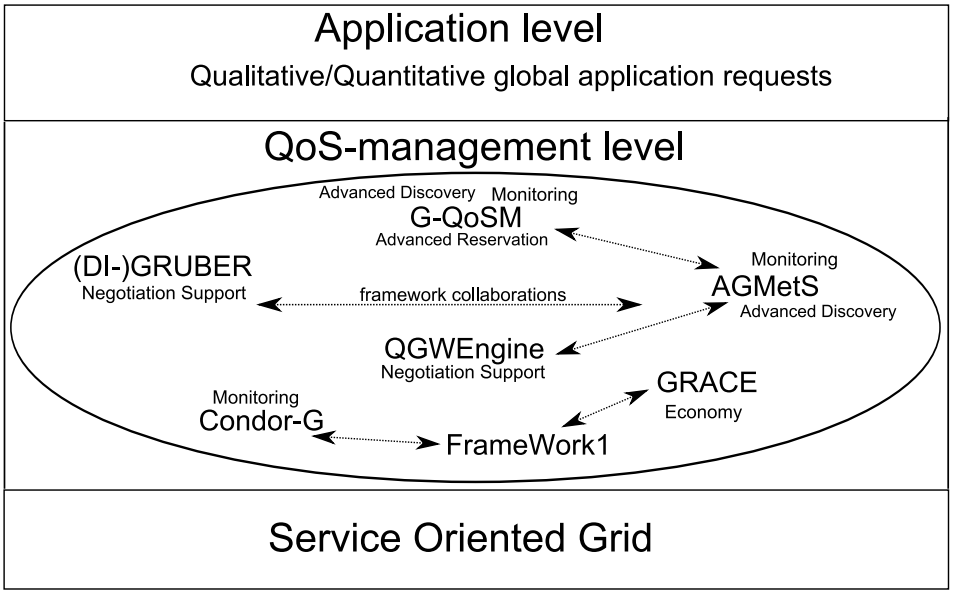

Figure 1.12: The QoS-management layer.

The **QoS-management layer** (Fig. 1.12 is an abstraction layer that stands between application layer and Grid layer, in order to define a level for explicitly providing support and functionalities to QoS over Grid. It has the aim to constitute a universal interface for both applications (and respective users for QoS requests) and Grid resources (and their owners for offering QoS over resources).

More specifically, it would be the reference level for managing QoS over Grids. Moreover, it aims to constitute a common starting point from which to define and implement QoS frameworks, considering that the previous analysis of current QoS architectures has shown too many different (and, in some case, contradictory) perspectives in studying QoS-support issue.

For this, in my proposal the QoS-management layer is composed by a set of **collaborative component frameworks**. Different frameworks or QoS-architectures compose the QoS-management layer so that high level applications could use functionalities from some of them, depending on the application specific QoS needs. In particular, QoS support is provided as a set of services (Advanced Discovery, Reservation, etc.) offered by the different frameworks.

In order to provide enhanced QoS support to applications (i.e. offering the whole set of QoS support requested by a real application), frameworks can be collaborative, i.e. they can combine single QoS features to grant support for all the QoS services required by a high level application. In this way, it become unnecessary to have a specialized framework for supporting QoS of a given class of applications. On the contrary, few frameworks could

support many kinds of applications by combining their features.

This would stimulate the definition of a framework specialized in managing a particular kind of QoS that interacts with other frameworks to obtain advanced QoS support as a *composition* of single QoS services, similarly to the idea of a common Service Oriented Architecture like current Service Grid.

The definition of QoS-management layer specifications and of interactions between application and Grid levels are currently in progress and would be the focus of my current and future works.

1.9 Final considerations

The analysis of both prominent architectures for supporting QoS over Grid and the evaluation of QoS requirements from real applications have taken to some considerations regarding a potentially efficient management of QoS on Grid.

First, QoS-management is a complex issue regarding different aspects in the whole set of operations on the Grid, involving resource provisioning and management. In particular, studying real application needs, I have shown that current approaches are able to provide efficient support only on a well defined subset of the required functionalities, so that a comprehensive support is theoretically reachable only merging subsets of functionalities belonging to the different approaches.

Secondly, since QoS requirements are becoming more and more strict (from early simple quantitative requirements to current emerging qualitative ones), a proper abstraction layer for managing QoS in a specialized way has to be considered explicitly (the *QoS-management level*). In fact, Service Oriented Grid middlewares cannot be able to manage all possible QoS requirements at Grid level; this would make middleware too complex, making hard to manage easily best effort resource provisioning that remains the basic aim of Grid Computing; for this, the explicit management of QoS both from resource owners (those who offer resources with QoS) and final users (that require QoS constraints for their final jobs) should be managed at a proper abstraction layer over Grid, without being invasive in Grid Computing basic functionalities.

On the other hand, I noticed that even if general-purpose Grids should remain free of QoS management, it is necessary for an efficient QoS support that owners interested in offering QoS on their resources explicitly give QoS guarantees, through suitable frameworks at QoS-management layer. In this context, the direct offer by owner has two consequences: first, the owner controls directly the offered resource, so it is able both to know which level of QoS can offer and to provide support to QoS-management layer frameworks.

As a general final remark on QoS management, I point out that even if the number of classes of applications increased thanks to the exploiting of QoS management and Economic Grid, nevertheless some classes of applications still remain "unmanaged" on Grid platforms, even if they could be suitable for the platform itself, exploiting the resources it offers. More specifically, applications with *strict time constraints* like Soft Real-Time ones are not still managed by current QoS architecture proposals, because of their inability to manage efficiently typical Grid operations under a time perspective. For instance, in proposed approaches, discovery, negotiation and execution are not time oriented nor an upper bound can be requested for every operation (e.g. Urgent Computing supports deadline in execution but only on previously selected resources, not dynamically and *on demand*). For this, the introduction of time constraints management in a QoS-compliant Grid architecture should be a further step that need to be done in QoS-framework evolution.

To this purpose, in the following chapter I present SoRTGrid, a framework for managing Grid operations under time-oriented constraints. SoRTGrid has been designed starting from the analysis of current QoS state of art and constraints of the Real-Time paradigm investigated here.

Chapter 2

The SoRTGrid Framework

The analysis made in the previous chapter has been the starting point in defining a proper framework, SoRTGrid (Soft Real-Time Grid) for supporting the management of time constraints on Grid. Before presenting SoRTGrid, it is important to fix a model for jobs with time constraints which can be executed on SoRTGrid, and briefly analyze the original features of this approach in comparison to previous analyzed QoS-compliant proposals.

2.1 A model for Soft Real-Time on Grid

In this section, I define a model for Soft Real-Time Systems (SRTS model) that will be used as a base for managing the problem on Grid. First of all, it is important to define a Soft Real-Time system in terms of simple jobs with deadline.

Definition 2.1.1. ***Soft Real-Time System.*** *A SRTS is composed by a set of Soft Real-Time jobs ($\{J_n\}$) that need to be executed within a given deadline and a set of resource bids ($\{R_m\}$) that grants the access to physical resources where the jobs can execute. I indicate a SRTS in this way: $SRTS = \{\{J_n\}_{n\in\mathbb{N}}, \{R_m\}_{m\in\mathbb{N}}\}$.*

In general, the set of jobs represents the task of a given set of Users; for the purpose of this thesis, it is not important how many users are related with a given SRTS; for this reason and for the sake of simplicity I suppose that an SRTS regards a single user only. On the other hand, the bids are offered by different resource Owners and the set of bids corresponds, as remarked, to the set of resources accessible by the user jobs.

A simple example of a SRTS is a video streaming system where any single soft Real-Time job can be a single frame or a group of frames to be elaborated. I present separately the

job and the resource sets characterization.

2.1.1 Jobs in a SRTS

In a SRTS, each Soft Real-Time job corresponds to (a set of) operations proper of the system. Any job is activated at a given instant. When activated, the job must complete within a maximum time, in order to be considered successfully executed. In general, a SRT job can have proper resource requirements for executing (e.g. CPU, RAM, disk space), a predictable length and unpredictable activation moment.

Before defining the characteristics of the SRT jobs considered in this thesis, some definitions are introduced.

Definition 2.1.2. ***Computational job.*** *A computational job is a job that has mainly computational requirements for executing. This means that it has no particular requirements in terms of RAM, disk space, network bandwidth and so on. In this context, a computational job is a job with only computational requirements.*

Definition 2.1.3. ***Deterministic job.*** *A job is deterministic if it is possible to appraise a maximum duration in its execution. In general, such duration is expressed in term of Million of Instructions (MI) the job uses for completing, in the worst case.*

I use Million Instructions as a duration counter because it is enough generic and currently widely used in high abstraction environments like the Grid one [SCV+08].

Definition 2.1.4. ***Job with unpredictable activation time.*** *A job with an unpredictable activation time is such that it is not possible to estimate the instant at which the job is activated.*

When a job is activated, it has to be dispatched on resources for executing. There must be at least a sufficient resource bid when the job is activated in order to complete the job properly.

I provide here a definition of the Soft Real-Time jobs I consider.

Definition 2.1.5. ***Soft Real-Time job.*** *A Soft Real-Time (SRT) job is a computational and deterministic job with unpredictable activation time, defined by a triple:*
$J_n =< MI_n, d_n, E_n >$
where:

- MI_n *is the length of the job in the worst case, in Million Instructions;*
- d_n *is the amount of time available for completing the job after activation;*

- *E_n is the instant at which the job is activated (unknown and unpredictable).*

From such parameters it is possible to obtain the absolute deadline of the job: $DL_n = E_n + d_n$. While the first two parameters are well-know and define the job, the third value is known as soon as the job is effectively activated; so, the deadline for the job completion is known only after its activation.

In a general-purpose SRTS, depending on the system behavior and on the way the time-constrained activities of the SRTS are divided into single jobs, jobs can activate at any instant and in any order. However, in order to reduce the complexity of the problem, I made the following assumptions:

Definition 2.1.6. ***Fixed set of jobs.*** *The set of jobs is defined once and before the SRTS runs, i.e. the number and the characteristics of the jobs do not change during execution.*

Definition 2.1.7. ***Sequential Jobs.*** *On a running SRTS, at every instant there is one activated job at most. This means that: $DL_n \leq E_{n+1}\ \forall n \in \mathbb{N}$*

In this modeling of the set of jobs, it is clear that the set has a maximum size when the SRTS begins its execution. A job element is removed when the job activates. The set of jobs could be even infinite (e.g. for modeling SRTS that are working indefinitely).

2.1.2 Resource Bids in a SRTS

A SRTS executes dispatching jobs on proper resources when they activate. In a time-oriented context, it is important to dispatch an activated job on a resource that is sufficient to complete the job within its deadline. Resources can be of any kind (CPU, RAM, disk storage), depending on requisites of the jobs.

In order to maintain consistency with the previous definition of computational jobs, I consider here only computational resources, that provide an amount of computational power expressed in MIPS (Million Instruction per Second). From a time constraint perspective, beyond the amount of MIPI provided, it is important the period of time in which a resource is offered by the resource owner. For this, the main characteristic of a resource owner for a SRTS is the offer of a bid, that grants the access to the resource for the offered time. Because SRTSs are limited to computational resource, here a definition of Computational Resource Bid that constitutes a SRTS is provided:

Definition 2.1.8. ***Computational Resource Bid.*** *A computational resource bid is defined by the couple*
$R_m =< MIPS_m, Exp_m >$
where:

- *$MIPS_m$ is the amount of Million Instruction per Second offered by the resource;*
- *Exp_m is the absolute expiration time for the computational power provisioning.*

On a running SRTS, the set of resource bids represents, at every moment, the resources on which it is possible to dispatch a job activated in that moment. When a resource bid reaches the expiration, it is removed from the set (i.e. the corresponding resource cannot be accessed). Other resource bids could be acquired and inserted in the set, depending on the resource scenario the SRTS works on.

The set of resource bids has a different nature from the set of jobs. In fact, while the set of jobs can only decrease during execution, the set of resources varies continuously depending on the requirements of the various jobs that are activated, or on the availability of resources that can be obtained.

The management of the resource bid set is the objective of the SRTS user, because its first aim is to possess, at any moment, enough resources to execute an activated job within its deadline.

Under this perspective, it is important to understand that, depending on the physical or logical resource scenario, the activity of the SRTS user could be simpler or trickier.

For instance, an embedded SRTS has a single resource. Such resource provides a fixed amount of MIPS and is always available. A different case is when the SRTS works over a local but general-purpose architecture like a single machine (single core), an homogeneous cluster or an heterogeneous cluster. On a single machine the access to the resource is likewise granted but can have a finite boundary (e.g. when the CPU is provided to another application in a typical multitasking system). On clusters, the maximum number of resources is well-known while the amount of MIPS can be identical for every computational resource (homogeneous cluster) or different (heterogeneous cluster). All of this scenarios do not require the SRTS to discover resources, because resources are fixed and well-known. In this sense, the activity of the SRTS is to select the best fitting resource from the set when the job activates (if there is one) and dispatch the job on the resource.

Moreover, a generic distributed system (e.g. a set of well-defined single machines, homogeneous and heterogeneous clusters) is different from a heterogeneous cluster because it can deal with different network bandwidth and many communication issues between single machines and clusters. Notwithstanding this, because I am dealing only with computational power, I can approximate a distributed system with a heterogeneous cluster (and an homogeneous cluster with a set of n equivalent single machines). However, a significant difference between a distributed system and a single big heterogeneous cluster is that in a distributed system physical resources can belong to a fixed and well known set of different owners, so the activity of the SRTS could require little complications due to performing requests for acquiring resources from such owners.

Although the presented examples are different, there is a common characteristic. In all cases, it is possible to upper-bound the number of elements in the resource set (i.e. the maximum number of physical resources that the resource set can contain) and therefore the kind and the maximum number of bids can be appraised.

The situation changes when dealing with Grid Computing where the impossibility to have total and fixed knowledge of the resources make impossible to define the maximum cardinality of the resource set that can vary largely during time, depending on the dynamism of the VOs (resources, users, and policies). In this sense, even owners of resources changes and must be firstly discovered in identity and in the resource provision they offer. For this, on Grid Computing the SRTS activity for acquiring resources (and then building up the resource set) is trickier than in previous case, because a large set of activities from discovery, to negotiation and selection has to be made in order to add resources to the set, without any guarantees or fixed knowledge on availability and nature of resources.

In the following table, I point out which assumptions can be made on the set of resource bids when dealing with the different considered underlying resource scenarios.

	Cardinality of $\{R_m\}$	Range of MIPS <MIPS, exp>	Range of EXP <mips, EXP>	Operations on resources
Embedded System	1 (a single resource ever)	fixed amount	Exp=infinite (dedicated resource ever available)	execution
Single Machine	1 (a single resource ever)	fixed amount	[0, T] (timeslot in a multitasking architecture)	dispatching execution
Homogeneous Cluster	[0,N] (N maximum number of processing unit)	fixed amount (identical MIPS of any processin unit)	[0, T] (T maximum time known and fixed)	selection, dispatching execution
Heterogenous Cluster	[0,N] (N maximum number of processing units)	[min,max] (fixed range of available processing units)	[0, T] (T maximum time known and fixed)	selection, dispatching execution
Distributed System	[0,N] (N maximum number of processing units)	[min,max] (fixed range of available processing units)	[0, T] (T maximum time known and fixed)	negotiation, selection, dispatching, execution
Grid Computing	[0,??] (maximum number ever changing and unrecognizable)	[??, ??] (dynamic changes in resource availability and amount)	[??, ??] (Individual owners with possible dynamic behavior and choices)	discovery, negotiation, selection, dispatching execution

Figure 2.1: Characteristics of $\{R_m\}$ set on different resource scenarios.

From the analysis of the table, it appears clear that Grid Computing introduces a non determinism level that needs to be considered in order to efficiently manage time constraints.

The problem of merging the model on the Grid will be analyzed later.

2.1.3 Matchmaking between jobs and resource

The aim of the SRTS user is to dispatch the activated job on resources sufficient for meeting the computational and time requirements of the jobs. In a general-purpose context, a job can have a sequential, parallel, interactive or batch behavior, and thus dispatched on any number of resources in the pool of acquired ones, in the bid resource set. For the sake of simplification, in my model I made the following assumption:

Definition 2.1.9. ***Single resource dispatch.*** *An activated job is dispatched at most on a single resource.*

This assumption reduces the complexity of the model and its evaluation but has the consequence that it is not possible to meet the deadline by dispatching the job on different resources. For this, the job is completed if there is a resource that it is sufficient by itself.

Under this assumption, I define here the concept of sufficiency for a given resource in the resource set.

Definition 2.1.10. ***Sufficiency of a resource bid.*** *Given an activated job* $J_n =< MI_n, d_n, E_n >$ *with deadline* $DL_n = E_n + d_n$*, a resource bid* $R_m =< MIPS_m, Exp_m >$ *is sufficient if and only if*
$MI_n \leq MIPS_m * (Min(Exp_m, DL_n) - E_n))$

I provide here a simply demonstration of the provided formula. First of all, I distinguish two possibilities in evaluating the sufficiency of a resource for a given job activated at current instant E_n:

- $Exp_m < DL_n$. In this case, the resource can offer its $MIPS_m$ for at most $(Exp_m - E_n)$ so the maximum amount of MIPS provided is defined by $MIPS_m * (Exp_m - E_n)$.
- $DL_n \leq Exp_m$. Here, the resource can provide at most $MIPS_m$ until the deadline (i.e. for $(DL_n - E_n)$). In fact, the provision is useless after the deadline expiration and further execution are useless. For this, the total amount provided by the resource is given by $MIPS_m * (DL_n - E_n)$.

From this, I evince that, in evaluating the sufficiency of a resource, the total amount of effective computation depends on the first event that happens between the deadline of the job and the expiration of the resource. Moreover, if the total amount is sufficient for covering the MI_n requirements of the job, than the resource is sufficient, otherwise the

resource is not able to fulfill the job requirements before the deadline.
From this consideration I obtain that the resource is sufficient if and only if
$MI_n \leq MIPS_m * (Min(Exp_m, DL_n) - E_n))$.

The estimation of the sufficiency of a resource is important when dealing with SRTSs that are predictable. Predictability is a fundamental hypothesis in the context I deal with in this thesis.

Definition 2.1.11. ***Predictability***. *Predictability is the degree to which a correct prediction or forecast of a system's state can be made either qualitatively or quantitatively.*

This general definition, applied to my model of SRTS, means that when a job is activated it must be possible to define a probability that the job could correctly complete within its deadline, considering the sufficiency of the resources belonging to the SRTS. From such evaluation, the SRTS could decide to execute the job or to drop it.

To this regard, I make a further assumption; for now, I do not take into account any algorithm for managing different behaviors of the system depending on the probability value obtained. On the other hand, I consider a single behavior of the SRTS user that dispatch a job only if there is the certainty of a correct completion (i.e. if there are sufficient resources, on a fault-free system). More specifically I make the following assumption:

Definition 2.1.12. ***Certain execution of Jobs.*** *If in the instant E_n of the activation of a job $J_n \in \{J_n\}$, the evaluation of the bids in $\{R_m\}$ provides at least a sufficient resource, than the job is executed and complete correctly within the deadline; otherwise, the job requirements are considered unsatisfiable and the job is dropped.*

At this point, it is possible to define a metric of the behavior of the system that I am interested in minimize with my approach when merging the model on the Grid.

Definition 2.1.13. ***(Average) Degrade of the SRTS***. *On a running SRTS, the (average) degrade of the system is given by the number of dropped jobs on the total activated jobs:*

$Deg_{SRTS} = \frac{Job_{dropped}}{Job_{activated}}$

The degrade of the system can be calculated at the end of the SRTS execution (for SRTSs with a limited number of jobs) or in any other moment. In the latter case, the calculation of the degrade degree provides an evaluation of the trend of the system until such moment.

The reduction of the degrade degree on a Grid environment is the main aim of the architecture I propose in this thesis.

2.2 The SRTSs on the Grid

The model presented can be implemented on different resource scenarios. Anyway, the efficiency of the system, inversely dependent on the degrade degree, is strictly tied up with the ability of the system to continuously have enough resources in the resource set so that any activation of the job could come to a dispatch.

Under this perspective, it is clear that the core aspect for the SRTS, in my model, is the acquisition of resources for their job, independently of the underlying resource scenario. However, depending on the characteristic of the scenario, the acquisition of resources can be differently tricky. For this, it is important to analyze the Grid paradigm from this point of view, exploiting which are the differences with a general distributed system.

2.2.1 Acquisition of resources: Grid Computing vs. Distributed System

Fig. 2.1 underlines that when merging the model on a distributed system, more assumptions are possible on the resource set than when dealing with Grid Computing. This is due to the fact that the Grid scenario is more dynamic and unpredictable from a resource point of view that a distributed (or local) system. Because Grid Computing can be considered as a meta-distributed level that gathers different distributed systems under a shared platform, I analyze here the differences between a distributed system and Grid Computing for acquiring resources, pointing out how such issue is much more complicated on Grid.

First, in a distributed system, the set of resources is fixed and well-known. This allows any SRTS to have a *total knowledge* on the set of possible resources to acquire. On the other hand, as remarked in chapter 1, Grid can be a very large distributed system, where the total knowledge is a wrong assumption. For this, any SRTS and, in general, any user on the Grid, has a knowledge of only a portion of the Grid. Besides, the knowledge must be build up through proper discovery activities, based on queries to proper index services. Anyway, all *knowledges* are different between SRTS, so it is important to check the *best* index service in order to find the most suitable resources, but it is not possible to know them previously, before the SRTS starts running on the Grid. For this, such knowledge is both *incremental* and *dynamic*. It is incremental because, when a user or SRTS enters the Grid, it needs some time to build up a knowledge on resources; moreover it is dynamic because the resources on the Grid change, so the discovery activity must be carried out continuously in order to keep it updated.

Another difference is that, in a distributed system, there is generally a given set of owners for the whole set of resources, with static policies for accessing them. For this, resources are fixed as the requirements for use them. On Grid, there are many owners sharing resources,

so identical resources but with different usability constraints can be offered. For instance, if the access to the resource requires a cost for the SRTS, in a distributed system a given resource is provided at a given fixed price, while on Grid there can be identical resources (in term of computational power and expiration) but with different prices, depending on choices made by the single owners. From a resource management point of view, there can be different policies on the same resources, and owners are free to change such policies, while policies in a distributed system are fixed and can be known soon after the system becomes online.

In a best effort context, the previous differences can have few consequences since no constraints on execution are necessary. On the other hand, when dealing with QoS, especially with time constraints on execution, such differences complicate the issue of acquiring suitable resources. Moreover, as remarked in the previous chapter, the Grid platform "as is" is not suitable for managing such constraints; a proper framework for supporting features not offered by Grid middleware, in order to simplify to deal with the Grid dynamism, making it as far as possible similar to a distributed system, is very useful from the SRTS (or general end-user) point of view.

2.2.2 Guidelines for defining SoRTGrid

Before presenting the framework, I make some final assumptions for merging the model on Grid. First, I assume that *any SRTS is managed on the Grid by a proper Grid user* who knows the set of jobs defining the system. The SoRTGrid framework has been defined starting from *four* architectural choices defined after the evaluation of the QoS state of art made in the previous chapter.

2.2.2.1 Independence of owner and user sides

The user has to operate in order to fill the resource set so that to any activation of a job can correspond a dispatch and an execution on a sufficient resource. Resources are managed by Grid owners, which have proper policies and define proper resource offers. First of all, the acquisition of resources requires a discovery activity started at the user side, in order to allow a user to find which owners offer resources that match its requirement. Moreover, the effective acquisition is obtained through interactions between owners and users. The framework must offer support for such activities, not implemented in current Grid middleware.

2.2.2.2 Precise resource offers

Resources are offered for a limited period of time, according with the model. Such choice is based on the consideration that it is not possible to offer unlimited guarantees on a Grid resource. In fact, by definition, a Grid resource is not a dedicated resource, but it is offered only during intervals of time when it is not used in the local physical organization. In a QoS context, this assumes a more relevant importance.

2.2.2.3 Economic Grid

Since resources must be offered so that they meet the user side requirements, my framework is supposed to work as an Economic Grid, because, as remarked, providing a gain to owners should stimulate them in offering QoS and resources following the requirements from the user side. In this way, a collaboration could reduce the randomicity of the resources in the Grid. On the other hand, competition reduces the natural heterogeneity of a Grid composed by different users and owners. For instance, the competition between owners (for providing better offers) tend to pull single owners towards common objectives (i.e. the user requirements), so resource offers are too focused, reducing the span of possible offers. For now, I do not manage explicitly the economic aspect of the Grid, but the whole architecture works on this assumption for granting an approach consistent with the current Grid trend.

2.2.2.4 QoS not invasive

The framework must provide direct support for QoS in a time-based context. It must allow to indicate expiration on offers and any other measure on QoS. Users use the framework to submit time constraints and owners make offers directly through the framework without any need of implementing the support to QoS and time constraints directly on the Grid middleware, that is required to be as general purpose as possible. This choice is made to preserve both the general-purpose nature of the Grid and the independence of owners and users. The first aim is unconditional, because Grid has to be maintained general-purpose in its conception; owners that aim to offer time-constraints on resources should have different ways other than the Grid middleware to make offers (a similar situation is valid for the user side). For the second reason, owners and users must always be free in their behavior so they choose to select the framework when they need/offer resources with time constraints; otherwise, they can use other frameworks (e.g. Fig. 1.12) or middleware services if, in a given moment, they have different needs.

2.3 Introduction to SoRTGrid

A framework aimed to be compliant with Soft Real-Time requirements must have the following features: 1) a resource discovery "in time"; 2) mechanisms to assure the availability of the required resources for the whole requested time-interval; 3) a mechanism of pre-reservation of resources to avoid inconsistencies during the negotiation phase. SoRTGrid tries to meet such requirements with a direct interaction between users and resource owners.
SoRTGrid constitutes a sort of facilitator between users and resource owners, a black box that provides different services allowing the efficient management of jobs with deadlines. In my context, a user is a Grid user that needs to execute a job with time constraints, while a resource owner is someone who can make available resources that he controls directly.
Both parts need some abilities to interact efficiently with SoRTGrid. The Grid user must be able to evaluate the length of his deterministic jobs so that the choice among the different resources could correctly take to a complete execution of the job within the requested time limit. The resource owner must control directly the resources it shares, without giving the control to any Grid metascheduler. In this way, the non-deterministic latency due to the metascheduler system and possible inconsistencies in the state of resources are avoided.
Users and owners are represented in SoRTGrid by proper autonomic agents (*User and Owner Agents*, respectively). Physical actors can interact with such entities to provide decisional policies or constraints, but are not directly involved in the offering, discovery, selection and negotiation of resources nor in execution activities. Discovery and selection of resources are carried out through the interaction of User and Owner Agents with SoRTGrid, while the subsequent negotiation and job execution are performed by direct interactions between such agents.
SoRTGrid makes available to Owner and User Agents a distributed shared data structure (Repository), similar in some aspects to the coordination model paradigm [Wel06].
The Owner Agent freely chooses which resources to offer and under which conditions; it produces a number of resource proposals (in the following SoRTBids) and writes them into the Repository. The User Agent provides a list of job requirements, the number of required SoRTBids, and a maximum time (*maxtime*) for the discovery activity.
SoRTGrid searches the published SoRTBids in the distributed Repository, and returns to the User Agent a set of SoRTBids matching the requirements. The discovery activity ends within maxtime. The User Agent chooses a subset of the retrieved SoRTBids, contacts the respective Owner Agents to negotiate them within the pre-reservation time, and finally execute its job on negotiated resources.
Moreover, SoRTGrid sends back to the Owner Agent feedbacks on the demand of its SoRTBids so that it can adapt its policies for the proposal of further SoRTBids to the User agent requests.
The architecture of SoRTGrid provides the following benefits:

- **Autonomy**. The two parts involved (users and owners) are independent in their actions and pursue different objectives. Users need to find resources that could grant them that a job submitted to the Grid could be correctly executed within the required deadline; owners require to have an efficient use of their resources under the terms and conditions they choose.

- **Adaptability**. The behavior of SoRTGrid depends on a set of parameters (e.g. the Grid size, the kind of resource requests and proposals respectively from the Grid user and resource owner side, allowed search time, number of neighbors). The appropriate evaluation of such parameters allows SoRTGrid to adapt to different Grid contexts.

- **Consistency**. Since SoRTGrid does not rely on metaschedulers but is based on a direct interaction between Owner and User Agents, it provides up-to-date information on resources that are effectively available. Moreover, it implements functionalities assuring that the provided information remains valid for a time interval sufficient to perform negotiation (timed pre-reservation), and that the same resources are not offered simultaneously to two different users (no overbooking).

- **Automation**. The use of agents eliminates direct interaction with human users from both resource and grid user sides. In this way a high level of automation of the parts involved is obtained and interaction time is reduced, according with the time-critical goal of SoRTGrid.

2.3.1 SRTSs and SoRTGrid

Considering the introduced global characteristics of SoRTGrid, the SRTS results easy to be merged on SoRTGrid framework. More specifically, the SRTS model on SoRTGrid is managed as follows:

- A User Agent (U_i) represents and manages a single Soft Real-Time System $SRTS_i = \{\{J_n\}_{n\in\mathbb{N}}, \{R_m\}_{m\in\mathbb{N}}\}$. The User Agent, when entering the Grid, knows only the job set composing the SRTS. The resource bids set is initially empty. The SRTS management activity concerning the User Agent regards all the phases (discovery, selection, negotiation, acquisition of resources) in order to fulfill the requirements of its job. More specifically, the User Agent has to acquire and have continuously available sufficient bids in the corresponding set in order to be able to execute the maximum number of activated job, reducing the degrade degree of its SRTS.

- An Owner Agent (O_j) possesses and directly controls a set of physical resources. It offers such resources in the form of *bids*, that are coherent with the description

of resource bids made in the SRTS model. In fact, the resource bid (in the following SoRTBid) has the form $SoRTBid_k =< MIPS_k, Exp_k >$ and grants the access to a certain physical resource. From a SoRTGrid point of view, in any moment the Owner Agent is characterized by its set of active (i.e. not expired) SoRTBids $\{SoRTBid_k\}_{k \in \mathbb{N}}$ published through SoRTGrid.

In this scenario, User Agents operate for discovering Owner Agents and their SoRTBids through a proper time-oriented architecture aimed at reducing latencies and maximizing both sides satisfaction. Once a User selects and acquires a given SoRTBid, it adds the bid in the pool of the possessed ones ($\{R_m\}$).

In the following section the architecture of SoRTGrid and provided services are analyzed.

2.4 The SoRTGrid Architecture

SoRTGrid is an overlay Grid network composed by a set of connected Facilitator Agents (Fig. 2.2) whose aim is to grant the required services both to the User and to the Owner Agents. Every Facilitator Agent is composed by two Grid Services [CCT05] (BidMan, DiPe) (Fig. 2.3) and interacts with the Grid middleware (e.g. Globus Toolkit 4 [Fos06]) to obtain basic Grid functionalities.
Virtual resources of the Grid are divided among different Facilitator Agents and every resource refers to a single Facilitator Agent only.
Every Facilitator Agent can work with every taxonomy of virtual resources. As a resource scenario example, I adopt here a classification I used in previous works [CCM+06], where resources are divided into Single Machines, Homogeneous Clusters and Heterogeneous Clusters (Fig. 2.3).
I consider, for further testing purposes, the Grid as constituted by the union of several sub-portions, each of which comprises only resources within an administrative domain (Physical Organization (PO)).
In the following, I assume, for the sake of simplicity, that there is a Facilitator Agent for every PO in SoRTGrid, and that the whole set of resources managed by an Owner Agent is related to a unique Facilitator Agent.

2.4.1 SoRTBids and Owner Agents

The aim of SoRTGrid is to provide sufficient support for jobs with time constraints. A key point is that a selected resource would be effectively available when the job needs to execute. As discussed, approaches relying on a shared middleware scheduler that controls virtual resources are unsuitable since there is not direct control on resources at lower level that

remain controlled by the local scheduler belonging to the Physical Organization. Indeed, since the owner manages directly the access to resources, the idleness of a certain virtual resource at the Grid level could not match with an effective availability at the physical level.
SoRTGrid resolves this possible inconsistency involving the resource owner and giving him the possibility to directly offer his resources to Grid users through a proper extension and customization of the mechanism of SLA Bids [RRB], the SoRTBids. In the SLA Bid approach, every owner decides in which way and how long to offer its resources, and publishes a number of SoRTBids.
In general, a SoRTBid can contain quantitative and qualitative information about one or more resources (number and frequency of processors, RAM and disk space, etc...). The set of resources is offered until an expiration time and for a time interval ΔT during which the execution of the job is guaranteed. The terms in which ΔT is offered define the *Quality of Service level* provided by the SoRTBid.
I choose to define and support different classes of QoS for different kinds of jobs. My aim is to provide a maximum guarantee level for jobs with critical deadlines and a lower level for those executions where missing a deadline does not imply serious consequences.

At present, SoRTGrid supports three different classes of QoS:

1. **Soft Real-Time Class**. The higher level of QoS; it guarantees the total availability of the resources for ΔT and no interruption of the job execution. This class is particularly suitable for interactive and time-constrained jobs.

2. **High Level Class**. The ΔT execution time is granted but the job can be interrupted. It is suitable for generic job with deadlines but not for interactive ones since the response time cannot be granted.

3. **Standard Class**. The ΔT execution time is granted only if jobs with higher QoS level do not run on the same resources. It is suitable for generic jobs without time constraints or such that the violation of a deadline does not take to serious consequences.

In my approach, the operations at owner's side regarding the definition and the publication of SoRTBids are performed by the Owner Agent. In practical, this can be an intelligent agent [fip97] or a proper application that implements logical rules or heuristics to create SoRTBids.
The Owner Agent has a dual interface, to the resource owner and to SoRTGrid (Fig. 2.3). On the one hand, Owner Agent accepts policies and parameters to adapt his decision model to the will of resource owner. On the other, it interacts with the BidMan service of the Facilitator Agent the resource belongs to; in this way it can publish and remove SoRTBids.

The use of the Owner Agent entity allows the automation of the SoRTBids production process, so that it could be quick, correct and independent of Grid user's behavior.

From resource side, SoRTGrid support total freedom in SoRTBid contents and three QoS levels. However, for the aims of this thesis I focused on the higher QoS level only, where I map the SoRTBids to the SRTS model. For this, a SoRTBid coherent with SRTS model ($SoRTBid_k =< MIPS_k, Exp_k >$) is a SoRTBid of Soft Real-Time class with only a quantitative attribute ($MIPS_k$) and an expiration time Exp_k such that: if t is the current instant, then $\Delta T = Exp_k - t$ and the resource is completely accessible and usable until Exp_k.

2.4.2 Facilitator Agent: BidMan Service

SoRTBids produced by Owner Agents must have visibility inside the Physical Organization and outside to the whole Grid. For this, Facilitator Agent has to receive and manage the set of SoRTBids referring to the resources it controls.
From my previous experience in deploying GEDA (Grid Explorer for Distributed Applications) [CDC+], I developed BidMan, a Grid Service for the management of SoRTBids, which constitutes one of the two Grid Services composing the Facilitator Agent.
BidMan creates and interacts with a shared repository (in the following, Local Repository) in which SoRTBids are contained. It allows an Owner Agent to upload new SoRTBids and to cancel previously uploaded ones. Moreover, it performs maintenance operations on the resident SoRTBids (i.e. removal of expired, cancelled or successfully negotiated bids). BidMan is the only Grid Service that has the rights to write on the Local Repository and that interacts with Owners Agents.
At this level, I do not provide any detail on the implementation of the Local Repository; for the focus of the thesis, it suffices to assume that it is addressable and accessible by the whole set of Facilitator Agents composing SoRTGrid.

Functionalities of BidMan

BidMan has been designed as a *singleton* Grid Service. This means that there is a single instance of BidMan that receives the offers from all the Owner Agents of the Physical Organization (Fig. 2.3). Every Owner Agent performs its task through a set of functionalities offered by BidMan Service:

- **SoRTBids upload**. It allows an Owner Agent to upload SoRTBids to the Local Repository. It returns a *unique ID* assigned by BidMan to every correctly uploaded SoRTBid. The ID is valid as long as the bid expires.

- **SoRTBids removal**. It allows to remove a not yet expired SoRTBid from the Local Repository. The removal is possible only if the bid is not being negotiated or reserved due to a previous discovery query. Otherwise, an error has to be reported to the Owner Agent.
- **Local Repository localization**. It returns the location of the Local Repository to the DiPe Grid Service for reading operations.
- **Local Repository state**. It returns up-to-date statistics on the Local Repository state like number of resident SoRTBids, pre-selected SoRTBids, free SoRTBids ad so on.

Besides the public functionalities, BidMan performs maintenance activities on the Local Repository like removing expired SoRTBids and freeing their *ID*.

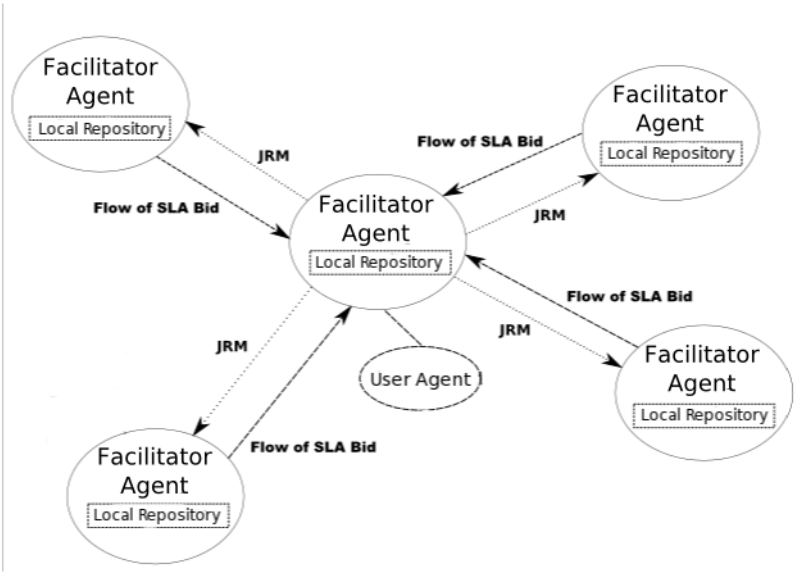

Figure 2.2: SoRTGrid as a set of interacting Facilitator Agents.

2.4.3 User Agents and Job Requirements Manifests

As remarked, every Grid user is represented in SoRTGrid by an User Agent.
The User Agent belongs to the Facilitator Agent to which belongs the respective Grid user. It receives and manages user's job requests, performing mainly three functionalities in user's stead:

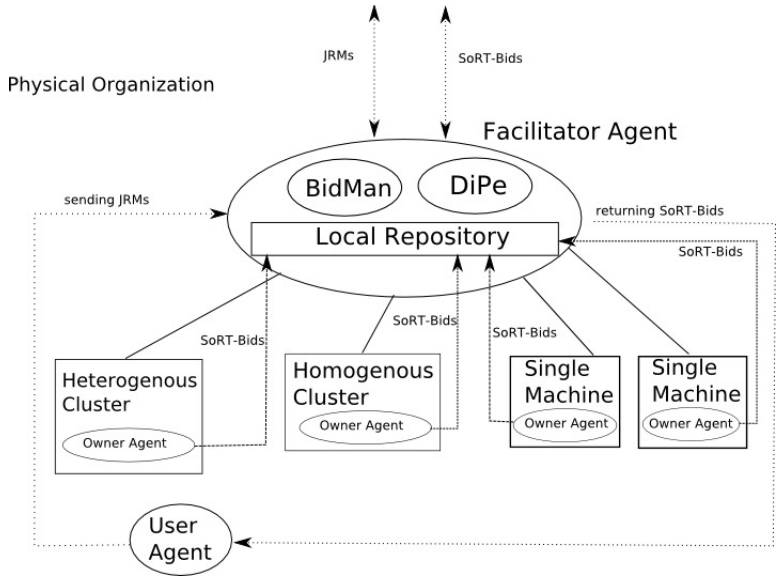

Figure 2.3: The Facilitator Agent composed by BidMan and DiPe Grid Services. It manages resource queries from User Agents and the access to the Local Repository.

- **Job requirements appraisal**. The User Agent receives jobs to execute and deadlines from the Grid user. From such information, it must be able to evaluate job execution time and on which resources; then it creates a proper requirements document (Job Requirements Manifest, in the following JRM).
- **Resource discovery and selection**. It uses the JRM to interact with Facilitator Agents and retrieves a set of suitable resources.
- **Job negotiation and execution**. It selects some resources from those returned by the previous phase and negotiates them directly with the respective Owner Agents. After a successful negotiation, it executes the job on such resources under the stated terms and conditions.

Because of the activity performed by the User Agent, the Grid user is not directly involved in requirements definition, resource selection and negotiation, and the whole activity is performed in an automatic way, perfectly according to Grid philosophy [Fos].

Let us now describe a general JRM document; it is composed of various fields:

- ***Resource requirements***. Quantitative and qualitative information about requested resources.

- ***QoS Class***. The class of required QoS and related parameters.
- ***Expiration Threshold***. Resource discovery will consider only SoRTBids with an expiration subsequent to this value. In fact, the user is only interested in SoRTBids available at the time T_{subm} when the job will effectively need to be submitted, so the threshold must be greater than or equal to T_{subm}.
- ***Appraisal Threshold***. During the discovery phase, SoRTBids are evaluated through a *utility function* that provides an *appraisal value* on the suitability of the SoRTBid for that JRM. Bigger is the appraisal value, more suitable is the SoRTBid. The Appraisal Threshold indicates the minimum appraisal such that the Bid could be considered *valid* for the JRM.
- ***Utility function parameters***. Values that provide *weights* for different variables in the utility function used to appraise a SoRTBid. Depending on job nature and user's requests, some characteristics of the SoRTBid can be considered more important than others.

JRM are sent by the User Agent to the Facilitator Agent to perform a discovery of suitable SoRTBids. Differently from common Grid resource discovery, in this case the time is the main variable to take into account.

Coherently with the SRTS model, also at the user side I consider a particular kind of JRM (that is created by the User Agent after taking into account the SRTS jobs characteristics). For this, even if by default SoRTGrid does not fix any limitation in the contents of a JRM with the previous structure, I consider only a quantitative requirement (MI_n), the Soft Real-Time class of QoS, and Expiration Threshold at least of DL_n from the moment in which the JRM is sent to SoRTGrid for resource discovery. The Appraisal Threshold considered is only 100% because of the certainty assumption in the SRTS model (cfr. 2.1.12), while no utility function parameters are considered because I deal only with a single resource requirement.

2.5 The physiology of SoRTGrid

In this section, basic functionalities of SoRTGrid are presented. In particular, functionalities for providing a time-oriented support to User Agent discovery activities are shown. Negotiation activities between selected Owner Agents do not directly regards SoRTGrid architecture.

2.5.1 Resource Discovery in SoRTGrid

The aim of resource discovery in SoRTGrid is to retrieve, within a given discovery time, a prefixed number (N_{Bids}) of SoRTBids with an appraisal value equal or greater than the Appraisal Threshold in the JRM.
The greater the N_{Bids} value, the longer will result the time required for the discovery and more SoRTBids will be potentially returned; on the contrary, a lower value grants a quicker discovery and fewer results.

Generally, the choice of the N_{Bids} value depends on the deadline of jobs that the User Agent needs to execute but can also depend on particular policies of the User Agent itself. Because of the criticality of time in the Discovery process, the aim of the discovery cannot be the retrieval of the best N_{Bids} SoRTBids in the whole SoRTGrid; on the contrary, the final target is to find the first N_{Bids} suitable SoRTBids (i.e. fulfilling the JRM requirements), interacting with as few as possible outer Facilitator Agents.
For the User Agent, the discovery activity must be as quick and efficient as possible, while, on the SoRTGrid hand, it has to produce a reduced number of communications between Facilitator Agents.
Moreover, resource discovery must be modeled to allow the retrieving of the desired number of SoRTBids within the required discovery time limit. I propose here a discovery algorithm aimed to take into account all the previous considerations.
The algorithm comprises two phases, namely *local discovery* and *remote discovery*. The second phase is performed only if the first one does not reach a satisfying result. Resource discovery is performed by a single Grid Service (DiPe) that, together with BidMan, constitutes the Facilitator Agent (Fig. 2.4).

2.5.1.1 Facilitator Agent: DiPe Grid Service

DiPe - that stands for Discovery Performer - is a *factory* Grid Service that after a User Agent request starts the local discovery and, if necessary, the remote discovery. The factory nature of DiPe implies that a new instance of DiPe is created at every request, and is dedicated to that query resolution. A DiPe instance is created due to a call from a User Agent or from another DiPe (Fig. 2.4).
A User Agent request is composed by three parts: the JRM, the number of required SoRTBids that must satisfy the JRM (N_{Bids}), and the time limit within which the discovery activity must be completed ($maxtime$).
Following a User Agent request, the new instance of DiPe performs first of all a local discovery on the Local Repository of SoRTBids. If the local discovery provides the required number of satisfying SoRTBids, the results are returned to the User Agent and the DiPe instance terminates. Otherwise, a remote discovery is performed by the DiPe instance through an interaction with other DiPe services belonging to neighbor Facilitator Agents;

a new instance is created for every DiPe service involved in the remote discovery.

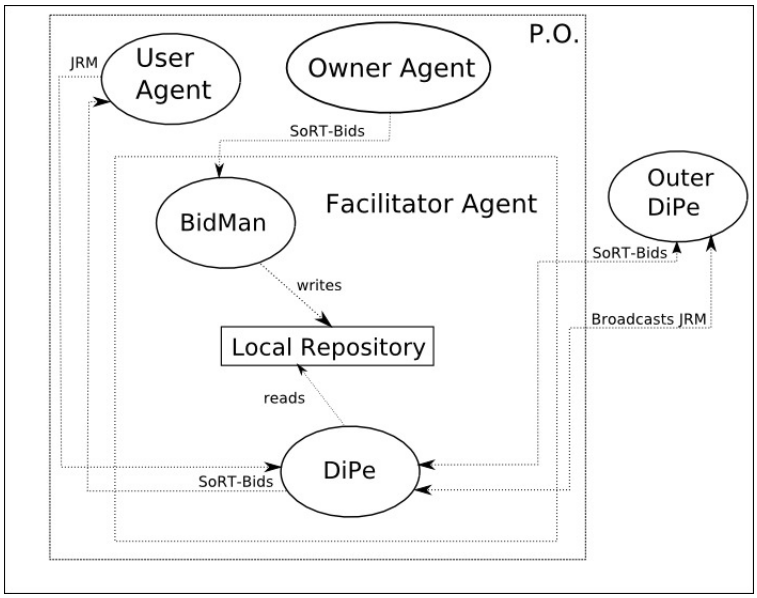

Figure 2.4: Interactions in SoRTGrid

Local and remote discovery will be presented in detail in the following. Here I present the functionalities of DiPe in order to define the functions and procedures that will be used in such discovery algorithms; the first functionality interfaces with the User Agent, whereas the others provide interaction with outer DiPe instances.

- **SoRTBids discovery**. It allows a User Agent to start a bid discovery submitting a User Agent request. DiPe performs a discovery activity that ends when N_{Bids} *suitable* SoRTBids are retrieved or *maxtime* is reached. In both cases, a group of SoRTBids is returned to the User Agent (the required number in the first case, a minor number in the second one). After the delivery of the SoRTBids, the instance is destroyed.
- **Remote SoRTBids request**. It is used by an outer instance of DiPe to require the search of SoRTBids in the Local Repository. It requires the URI of the requesting DiPe and the JRM. Selected SoRTBids are returned to the requesting DiPe through the next functionality.
- **Remote SoRTBids delivery**. It is used by an outer DiPe instance to deliver the SoRTBids it found.

The system automatically divides the maximum search time (*maxtime*) into two portions, namely t_{loc} (the maximum time allowed for local discovery) and t_{rem} (the maximum time

allowed for remote discovery). The way in which search time is divided can impact in SoRTGrid behavior; as a particular case I can also start local and remote search at the same time.

2.5.1.2 Local SoRTBid discovery

Local discovery starts after a call from a User Agent to the DiPe functionality *SoRTBids discovery*, creating a new local instance of DiPe that manages the whole discovery process. The local instance interacts with the Local Repository, reads and appraises every SoRTBid, defining incrementally a subset of SoRTBids whose appraisal value is greater than or equal to the Appraisal Threshold in the JRM. This process terminates when all the SoRTBids have been checked or t_{loc} has passed.
Two roads can then be followed, depending on the number of suitable SoRTBids found in the Local Search:

- $N_{loc} \geq N_{Bids}$. The required number of SoRTBids is reached. N_{Bids} SoRTBids are returned to the User Agent, the local DiPe instance and the discovery activity end. The returned SoRTBids are those with lower appraisal value. This choice depends on a Grid optimization issue: every selected SoRTBid satisfies the JRM, so it is not a good choice to return to the User Agent resource proposals that exceed a lot the real requirements, since it will take to the negotiation of resources that will not be totally used.
- $N_{loc} < N_{Bids}$. The local discovery has not found enough suitable SoRTBids, so a remote discovery will be performed in the remaining time t_{rem}.

2.5.1.3 Remote SoRTBid discovery

If the local discovery is not sufficient to satisfy SoRTBids request of the User Agent, then a discovery on other Facilitator Agents has to be performed.
The remote discovery is started by the *local DiPe instance* (i.e the one that interacts with the User Agent), and involves a set of *remote DiPe instances* that answer to the remote request. The algorithm works as follows:

- ***Initialization***. $N_{rem} = N_{Bids} - N_{loc}$ is the amount of suitable SoRTBids the remote discovery has to retrieve. $t_{rem} = maxtime - t_{loc}$ is the remaining time for the remote discovery.
- ***Local DiPe instance***. The local DiPe instance starts the remote discovery by using the *Remote SoRTBids request* functionality of *neighbor* Facilitator Agents. Afterwards, it waits until N_{rem} Bids are delivered or t_{rem} passed, and then terminates.

- ***Remote DiPe instance***. First of all, the DiPe Grid Service receiving the call checks if it has already answered to a previous request coming from the requesting DiPe instance for the same JRM. If so, it discards the request. Otherwise, it creates a new DiPe instance that performs a local discovery and tries to send back the set of suitable SoRTBids it has found through the *remote SoRTBid delivery* functionality of the requesting DiPe instance. If the requesting instance still exists, the SoRTBids are correctly delivered and the DiPe instance sends the request to a proper[1] subset of its neighbors by calling their *remote SoRTBids request* functionality and sending the JRM it previously received. In this way a new remote discovery step is started. On the contrary, if the requesting DiPe instance is no longer available, the broadcast is not performed.

- ***Termination***. The local DiPe instance that originated the remote discovery terminates when it receives the required number of SoRTBids (i.e. $\geq N_{Bids}$) or when t_{rem} expires. In the first case it returns back to the User Agent the first N_{Bids} with the lower appraisal value, whereas in the second the whole set of SoRTBids retrieved till expiration is returned.

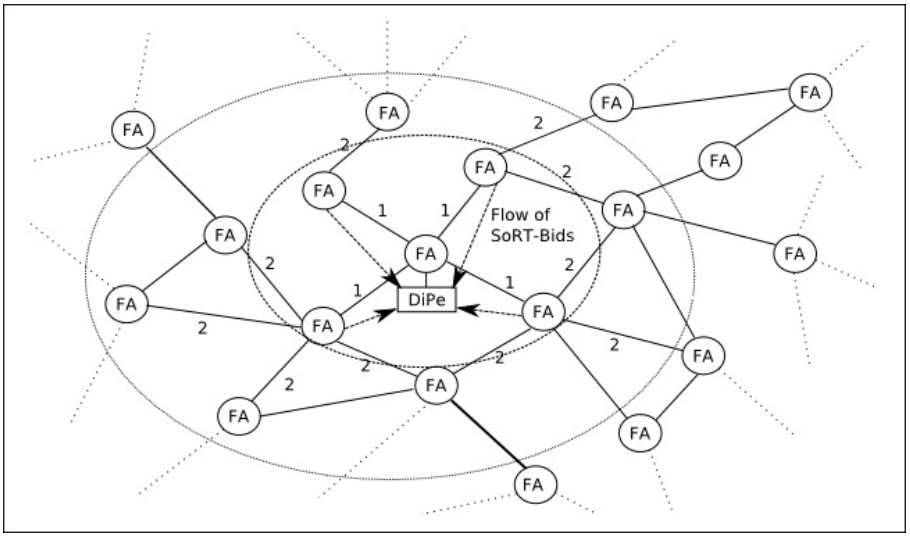

Figure 2.5: A two-steps remote discovery based on broadcast. The inner ellipse indicates the first broadcast, the outer one stands for the second step. A third broadcast is not forwarded because the requesting DiPe instance terminates before the end of the second step.

[1]The choice of the set of neighbors to which forward the request can be defined through different algorithms and based on several kind of information. This aspect will be expanded in the section dedicated to SoRTGrid implementation.

The main advantage of the remote discovery algorithm is that it follows a completely distributed approach in which every outer Facilitator Agent manages independently a local discovery and a SoRTBids delivery to the unique DiPe instance that originated the discovery activity. Such instance receives **concurrent flows of SoRTBids** from different portions (POs) of the SoRTGrid until the desired amount is retrieved.
Since the request is forwarded simultaneously to a number of neighbors, groups of local discoveries are performed in parallel, reducing the total discovery time. Moreover, the broadcast is forwarded to neighbors only if there is still need of SoRTBids (i.e. the request in the DiPe instance is still existing) and the same DiPe instance does not execute a second discovery for the same request; both these characteristics allow to avoid useless searches, improving the answer time.
Denoting with N_{FA} the number of Facilitator Agents, the number of steps in the remote discovery phase is bounded by $\log_{neigh} N_{FA}$, where $neigh$ is the average number of neighbors of the Facilitator Agents.

2.5.1.4 Overlay network between FAs

As shown in the previous sections, SoRTGrid consists of an indirect graph (see Fig. 2.5) where nodes are Facilitator Agents and edges are neighboring relationships (i.e. if A and B are directly connected it means that A is neighbor of B and vice versa).

The topology is built incrementally by subsequent entrances of Facilitator Agents. Initially, SoRTGrid is composed by a single Facilitator Agent; afterwards other Facilitator Agents can join SoRTGrid. Whenever a new Facilitator Agent enters or leaves SoRTGrid, a proper graph algorithm manages the topology, granting that the graph is completely connected.

In this thesis I have considered a single approach based on a Delaunay triangulation. It is clear that further algorithm can be used.

2.5.1.4.1 Labeled link and Delaunay Triangulation In this approach, FAs know each other by registering themselves using a proper Grid Service called PO Registration Service (PORS). Every FA registers itself during the start-up phase (i.e. when enters in SoRTGrid), and it receives back the addresses of its neighbors. The PORS organizes GEDAs as a Delaunay triangulation by associating to each instance two random coordinates within a fixed rectangle. This solution gives a degree of six for the triangulation vertexes and a good load balancing.

The creation of the triangulation is performed in an incremental way, as described in [GMCD08]. Periodically the PORS checks the state of the registered FAs, in order to update the triangulation if some instances are no more available. In this case, it sends to the FAs involved in the re-triangulation the updated list of the neighbors.

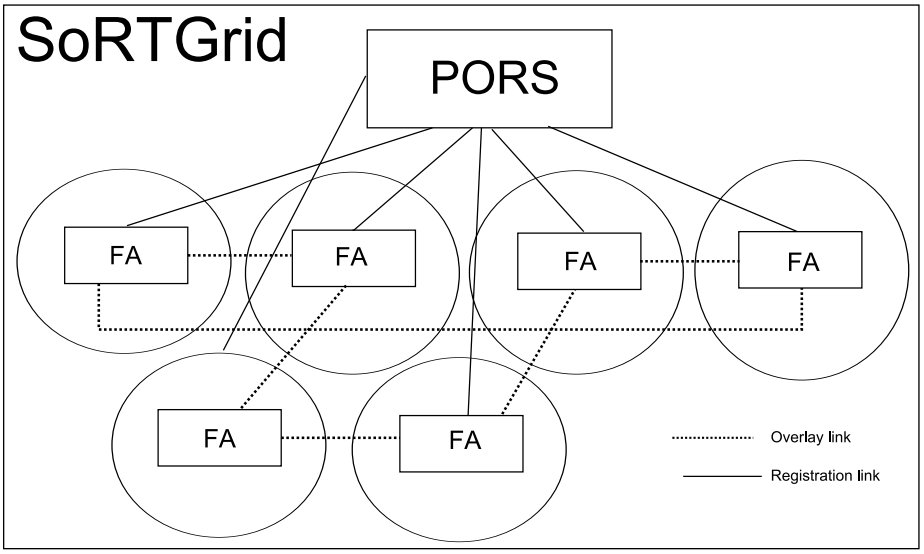

Figure 2.6: Building of overlay network of FAs.

This list is important for the selection of the resources for user requests. I avoided solutions based on too complex routing mechanisms. In fact, I have to consider that the parameters a user may specify for its job are fixed by the query language (the JRM syntax). For this reason each FA processes the information about the resources it represents and produces a table with entries of this type $< attribute; minvalue; maxvalue >$. For example, MIPS can represent the range of the MIPS available in a given moment within the Local Repository.

Each FA makes available its table to the first neighbors and it receives theirs. When it is queried by a user or by another FA through a DiPe instance, at first it performs the Local Discovery within its Local Repository. If it does not find N_{Bids} candidates it forward the query to a subset of neighbors satisfying the requirements, in a random way.

2.5.2 Notes on SoRTBids Negotiation and Production

It is important to avoid situations of inconsistency in the state and the management of SoRTBids during their life-cycle from the definition to the negotiation.
In the following I will point out some critical situations, proposing guidelines for their management in SoRTGrid.

2.5.2.1 Timed pre-reservation of resources

Resource retrieving in SoRTGrid differs from a typical Grid discovery because the negotiation and acquisition of resources is not mediated by any Grid metascheduler, but is performed through a direct interaction between the User Agent and the owner of the selected resource using the mechanism of *SLA-Negotiation*.
A possible inconsistency is that a certain SoRTBid selected by the User Agent could be not effectively available when the User Agent tries to perform a SLA-negotiation.
Such situation may occur in two cases:

- ***SoRTBid yet negotiated***. In the time interval between the return of the SoRTBid to the requesting User Agent and the start of SoRTBid Negotiation, the SoRTBid has been negotiated by another User Agent.
- ***SoRTBid removed by the Owner Agent***. Before the SLA-Negotiation, the owner removes from its local Repository the SoRTBid previously selected by the User Agent.

The way to avoid these inconsistencies is to adopt a *timed pre-reservation* on the SoRTBids returned by a discovery. For a certain amount of time (T_{PR}), the SoRTBids are reserved for the requesting User Agent: this means that during this pre-reservation interval they cannot be removed by the owner nor considered valid or returned for every other User Agent query.
In this way, the timed pre-reservation assures to the User Agent the effective negotiability of the SoRTBids obtained by a discovery.

2.5.2.2 SoRTBids overbooking and abuse management.

An Owner Agent can produce any number of SoRTBids for the same resource: this creates another inconsistency problem due to the fact that, at every instance, the total SoRTBid proposal can overcome the real capacity of the resource.
For instance, let us suppose that two *Soft Real-Time Class* SoRTBids offer the total availability of a CPU for two periods of time that overlap one another. Since it is impossible to grant the simultaneous availability of both SoRTBids, such situations must be avoided by the Owner Agent; moreover, BidMan performs a control activity on Owner Agent behavior checking the SoRTBids contained in the Local Repository.
For this, a set of rules and control policies has to be defined to grant both independence to the Owner Agent for SoRTBids production and a satisfying level of guarantee on SoRTBid negotiation for the User Agent.

In SoRTGrid, Facilitator Agents act as controllers, in order to label the Owner Agents that, due to incorrect or malicious overbooking policies take to not satisfy the contract with User Agent counterpart.

Chapter 3

SoRTGrid implementation and features

In this chapter, I present both the current implementation and features of SoRTGrid and a platform for managing computational jobs on Grid. The global architecture presented in the previous chapter has been implemented together with some other additional features for testing purpose. Under this perspective, SoRTGrid has an *open implementation* in the sense that the global framework and behavior are implemented (e.g. the DiPe, BidMan services, Local Repository), but specific behaviors of User and Owner Agents can be defined modularly through proper extensions, in order to test different artificial intelligences, policies and so on. Moreover, the current implementation of SoRTGrid is parametric and completely configurable in any part. The final aim is to provide both a platform able to test any behavior or possible scenario concerning the management of time-constraints on Grid and an effective framework able to work on a real Service Oriented Grid.

3.1 SoRTGrid standards and implementation guidelines

SoRTGrid has been implemented with the aim of building an effective framework, able to work with a general purpose Service Oriented Grid. Although I have been interested on Globus Toolkit 4, SoRTGrid is open to be adapted to further Service Oriented middleware.

Because an effective implementation of SoRTGrid could have been tricky to be tested directly on a Grid with enough resources and different scenarios, the development of SoRTGrid has been defined through two different steps. The first one regards the development of a first implementation of SoRTGrid that is effectively the global working framework but

with an interface with a Service Oriented Grid simulator (GridSim [BMA02]), in order to potentially test the SoRTGrid behavior on different Grid scenarios, even huge, otherwise impossible to test. The second regards the effective porting of SoRTGrid on a real Grid with Globus Toolkit 4. It is important to point out that the porting from GridSim to a real Grid is not an hard issue, and the further presentation of the implementation choices of SoRTGrid will prove this fact.

For this, in this thesis I concentrate on presenting a part of the first step that has been realized, namely how to implement a general architecture of SoRTGrid able to be ported both on a simulated (GridSim) or real (GT4) Grid. An analysis of the final part of the first step (i.e. the porting of the implemented architecture on GridSim) will be provided in the next chapter, dedicated to Grid simulation. The porting of current version of SoRTGrid to Globus Toolkit 4 is, as remarked, a simple development activity still in progress. For this, in the end of the chapter I provide a short presentation of this work-in-progress.

First of all, it is important to notice that SoRTGrid has been implemented using two basic programming technologies, namely Java and XML.

SoRTGrid has been implemented in Java for two main reasons:

- *Portability.* Java is a portable object oriented language. The portability of Java allows a program to be compiled once and to run on any Java Virtual Machine (JVM), independently of the actual operating system and architecture. For this, it is really suitable in a general purpose architecture like Grid.
- *Compatibility.* GridSim is a toolkit implemented in Java, defined by a set of Application Program Interfaces (APIs) that wrap basic Service Oriented Grid Structures as Java classes. The choice of developing SoRTGrid in Java makes simple the interface towards the GridSim toolkit. Moreover, Java applications are well supported by middleware like Globus Toolkit 4; this is important under the perspective of porting SoRTGrid to a real Grid.

For similar motivations, I have chosen to adopt XML as the format for managing documents concerning SoRTGrid. Indeed, XML can be easily extended and a large number of tools for parsing XML files are available. In SoRTGrid context, XML is extensible since it allows the user to define proper mark-up elements, making the document able to contain any kind of information easily shareable over networks and domains. Briefly, any XML file is based on a proper schema defining the structure and tags of a valid document based on that schema. In this way, it is simple both to change the schema for adding information and verifying the correctness of a given document and the compliancy with the model constraints. From a SoRTGrid perspective, the use of XML grants a simple merging of the SRTS model on SoRTGrid in XML based SoRTBids and JRMs, making an easier task to define constraints and controls over correctness of such documents. Moreover, SRTS model

currently considers only computational attributes, but its schemes are open to extensions in term of resources considered and QoS levels. Thus, the use of XML grants a quick implementation of these extensions through simple modification of SoRTBids and JRM schemes.

Java and XML are used for implementing two different aspects of SoRTGrid. The *architecture* and main components of SoRTGrid (e.g. Facilitator, User and Owner Agents) are implemented as Java classes, so that their interactions can be managed as a set of local or remote JAVA RMI calls between Java objects. On the other hand, SoRTGrid is made by an *Information component* realized through its documents (JRMs and SoRTBids) containing information exchanged between Agents during their activity. These documents are coded in XML.

These parts constitute two different abstraction layers (Fig. 3.1) in SoRTGrid, with different characteristics from an implementation point of view.

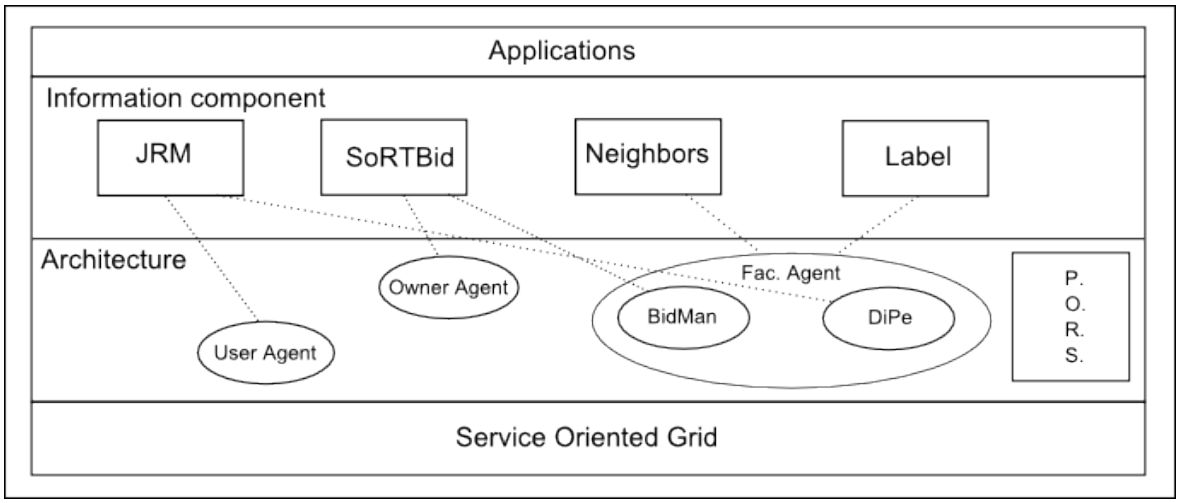

Figure 3.1: SoRTGrid abstraction layers.

The information component is tied up with the application and resource scenario so the documents definition must be enough general to provide support for as many application scenarios as possible. On the other hand, it is not a concern for this component the nature of the Grid (real or simulated) it operates on.

On the contrary, the architecture and its behavior is not dependent on the kind of jobs and resources considered and on the attributes used to describe them. In fact, SoRTGrid provides discovery and selection based on set of attributes, independently of the application context. Indeed, the architectural side depends on the underlying Grid middleware, so such implementation needs to adapt to services offered by the middleware. SoRTGrid works on a Service Oriented Grid middleware following OGSA specification, since it grants the

possibility to assume the provisioning of a minimal set of services by the middleware. The way in which such services are offered concerns the implementation of SoRTGrid architecture.

As remarked in the previous chapter, SoRTGrid is general-purpose and can be potentially adapted to any Grid middleware and can potentially manage any kind of computational job (with or without deadline). For this, SoRTGrid comprises XML schemes extensible to any job requirement and resource offer; in the same way, Agents are implemented in Java like classes that can be adapted to real or simulated middleware by proper extensions of the basic classes.

Depending on the application scenario to simulate and on the underlying middleware, XML and Java components of SoRTGrid must be differently adapted. Because the XML information component is strictly dependent on the application scenario, general XML schemes are adapted through proper simplifications (e.g. considering only needed tags and variables) to the application scenario requirements. On the other hand, while the functionalities and interactions of Java components are definitely implemented independently of the middleware (realizing the behavior of the framework), the interface with the underlying middleware has to be realized by implementing proper methods of the Java classes, so that their redefinition can grant support even to other OGSA-based middleware.

In this thesis, I am interested in modeling SRTS as the application scenario, and GridSim as a simulated Grid. Before presenting the current implementation of SoRTGrid that works under these assumptions, I first provide an introduction to the general implementation of the information components of SoRTGrid for computational jobs, and then describe how the general information documents (JRMs and SoRTBids) have been adapted to the SRTS scenario.

3.2 Defining the Information Component

SoRTGrid works over any possible set of resources and jobs; for this, it is not a SoRTGrid issue to know and check physical resources belonging to Owners, nor the evaluation of jobs executable of Users. In fact, SoRTGrid is not an execution environment but only a discovery one with services for granting interactions between users and owners: the knowledge on resources and jobs is provided to SoRTGrid indirectly through SoRTBids and JRMs.

For this, information contained in SoRTBids and JRMs has not to be a comprehensive description of the resource offered or the job to be executed; on the contrary, information provided in these documents must be sufficient for granting a proficient interaction between the two parts.

From an implementation perspective, this means that it is not an important issue how to implement resources and jobs, while it is strategic to provide a *good* implementation of both SoRTBids and JRMs.

3.2.1 Analysis of general JRM and SoRTBids

Concerning JRM and SoRTBids, I modeled the schema reported in Fig. 3.2.

```
- <JRM>
  - <JOBREQUIREMENTS>
      <MIPS>4</MIPS>
      <MAXEXECTIME>3000</MAXEXECTIME>
      ...
      <MINIMUMQOS>2</MINIMUMQOS>
    </JOBREQUIREMENTS>
    <VALUETHRESHOLD>65.0</VALUETHRESHOLD>
    <COSTTHRESHOLD>10</COSTTHRESHOLD>
    <MATCHINGBIDSNUM>2</MATCHINGBIDSNUM>
  - <DISCOVERYDEADLINE>
      <YEAR>2009</YEAR>
      <MONTH>2</MONTH>
      <DAY>6</DAY>
      <HOUR>12</HOUR>
      <MINUTE>20</MINUTE>
      <SECOND>30</SECOND>
    </DISCOVERYDEADLINE>
  - <EXECUTIONDEADLINE>
      <YEAR>2009</YEAR>
      <MONTH>2</MONTH>
      <DAY>6</DAY>
      <HOUR>12</HOUR>
      <MINUTE>35</MINUTE>
      <SECOND>45</SECOND>
    </EXECUTIONDEADLINE>
  - <STARTEXECUTION>
      <YEAR>2009</YEAR>
      <MONTH>2</MONTH>
      <DAY>6</DAY>
      <HOUR>12</HOUR>
      <MINUTE>22</MINUTE>
      <SECOND>30</SECOND>
    </STARTEXECUTION>
  - <ENDEXECUTION>
      <YEAR>2009</YEAR>
      <MONTH>2</MONTH>
      <DAY>6</DAY>
      <HOUR>12</HOUR>
      <MINUTE>30</MINUTE>
      <SECOND>30</SECOND>
    </ENDEXECUTION>
  </JRM>
```

```
- <SORTBID>
  - <RESOURCEATTRS>
      <QOSLEVEL>1</QOSLEVEL>
    - <CP>
        <MIPS>3.2</MIPS>
        <PERCENTAGE>70.0</PERCENTAGE>
      </CP>
      ...
    </RESOURCEATTRS>
  - <BEGIN>
      <YEARb>2009</YEARb>
      <MONTHb>2</MONTHb>
      <DAYb>1</DAYb>
      <HOURb>4</HOURb>
      <MINUTEb>0</MINUTEb>
      <SECONDb>0</SECONDb>
    </BEGIN>
  - <END>
      <YEARe>2009</YEARe>
      <MONTHe>2</MONTHe>
      <DAYe>1</DAYe>
      <HOURe>16</HOURe>
      <MINUTEe>0</MINUTEe>
      <SECONDe>0</SECONDe>
    </END>
  </SORTBID>
```

Figure 3.2: General schema for JRM and SoRTBids.

Starting from JRM, this kind of document contains a first part for attributes regarding the requirements of jobs (<*JOBREQUIREMENTS*>). In general, no assumptions are made

on the kind of attributes (qualitative or quantitative) supported. In this example, there are only computational requirements, expressed as an average execution frequency <*MIPS*> for at least a given duration <*MINEXECTIME*>. For any kind of quantitative attributes, a minimum level of QoS (<*MINIMUMQOS*> is defined, coherently with the modeling in 2.4.1; the QoS level defines the minimum conditions under which the quantitative attributes must be provided (e.g. Percentage of computational power that can be provided, with or without interruptions (Real-Time Class vs. Standard Class)).

Moreover, the first part contains the number of SoRTBids requested (<*MATCHINGBIDSNUM*>) and the threshold value (<*VALUETHRESHOLD*>, i.e. how much a SoRTBid needs to fulfill the JRM attributes for being considered *suitable* and then selected). Besides, since SoRTGrid can potentially (and it is supposed to) work over Economic Grid, the <*COSTTHRESHOLD*> parameter allows to define the maximum cost affordable for a sufficient SoRTBid.

Because SoRTGrid provides support for discovery, a deadline for the discovery activity (<*DISCOVERYDEADLINE*>) can be specified. From a SoRTGrid perspective, such deadline is hard; for this, SoRTGrid returns the maximum number of desired SoRTBids. When such deadline is reached, all the activities involving directly the framework are terminated and the selected SoRTBids are returned.

The execution deadline (<*EXECUTIONDEADLINE*>) regards both the framework and the Owner Agents side. From the framework side, it is used not as a fundamental parameter, but to discriminate in the selection of two matching SoRTBids (e.g. if a matching SoRTBid ends after the deadline it is selected the other one). From the Owner Agents side, it is used for acquire information on average duration of user side deadlines, for successive bid proposals.

Depending on the nature of the job, there is the possibility to define an interval of time during which the job can activate (<*STARTEXECUTION*> and <*ENDEXECUTION*>. Such information is mandatory: if those tags are defined, activation of jobs before and after the period is not considered as correct. In order to provide a clear meaning of this, it is important to consider the global scenario: Owner and User Agents interact to come to an agreement under the form of a Service Level Agreement. In this sense, the JRM contains all the contractual constraints belonging to the user side. On resource side, if a given resource is provided out of the contract constraints, the owner is considered incorrect (SoRTGrid, as remarked, comprises an abuse management system). In the same way, User Agents are evaluated to check if the information on jobs to execute is correct.

Like JRM, a SoRTBid provides a set of qualitative and/or quantitative attributes describing the (set of) resources offered (<*RESOURCEATTRS*>. Any quantitative attribute (in these case the Computational Power (<*CP*>) can be offered in a given percentage (<*PERCENTAGE*>), coherently with the given QoS level (<*QOSLEVEL*>). For instance,

in the Soft Real-Time class this means that the 50% of the resource is provided from the beginning of execution continuously; for Standard class the 50% is guaranteed on the period of time in which the resource is offered, while in Best Effort class the percentage is guaranteed only if other jobs with higher class are not running. Note that any quantitative attribute has a different level of QoS.

The second part of a SoRTBids contains data defining the period of time in which the resource is effectively accessible (i.e. during which the job can be dispatched on it and execute) under the conditions contained in the first part. This means that any SoRTBid, in general, does not grant a direct access once the SoRTBid is accepted.

3.2.1.1 Merging JRM and SoRTBids schemes on SRTS model

The given XML structures can be adapted in order to represent different kinds of jobs and resources. For example, a best effort job can be modeled on the JRM schema by uninstantiating all tags regarding time constraints (<*DEADLINE> and <*EXECUTION>) and fixing <QOSLEVEL> to 1. Moreover, a job with fixed activation can have the same values both in <STARTEXECUTION> and <ENDEXECUTION>.

On resource side, a typical best effort Grid resource, providing no guarantees on availability and execution, can be offered in a SoRTBid without instantiating <BEGIN> or <END>, and <QOSLEVEL> equal to 1. On the other hand, a Soft Real-Time resource offers <QOSLEVEL> equal to 3, 100% of <PERCENTAGE> and no <BEGIN> or <END> defined (always available at maximum quality, even if, as previously remarked, it is uncommon to assume availability of such dedicated resources on Grid).

In this context, merging the SRTS model on such schemes is quite simple. JRMs and SoRTBids representing SRTS are in the form shown in Fig. 3.3.

The JRM of a SRTS is composed of a couple of quantitative attributes (<MIPS> and <MAXEXECTIME>), indicating the average MIPS required and the period of time for which the computational power must be provided at least. From these attributes it is possible to obtain the worst case MI_n of a job J_n as a product of MIPS and MAXEXECTIME. The requested class of QoS is the Soft Real-Time one, while the <VALUETHRESHOLD> is, for assumptions made, fixed at 100%, considering only SoRTBids that totally match the job requirements. There are no constraints on <DISCOVERYDEADLINE> and <EXECUTIONDEADLINE>, leaving full freedom of defining them to the User Agent: it is clear that a later discovery deadline provides more time for discovering; this means a deeper search activity on the Grid and potentially more matching SoRTBids results. On the other hand, as the time limit for completing the job is defined (d_n) (and so the execution deadline is fixed), enlarging the discovery deadline takes to a reduction on the effective

Figure 3.3: JRM and SoRTBid schema for SRTS.

time for negotiating and acquiring the SoRTBids and for job execution. Such choices can be made from different perspectives by the single User Agent, depending on job nature and, in particular, consequences for the SRTS in case of the miss of the execution deadline.

Since jobs in SRTS have an unpredictable activation (cfr. 2.1.4), <*STARTEXECUTION*> and <*ENDEXECUTION*> tags are not considered, making the job potentially activable at every instant. As a last remark, I do not consider too short deadlines (e.g. in the order of seconds) for two reasons. First, shortest and more precise deadlines are important for Hard Real-Time, I do not consider them here nor they are affordable on a Grid context. Moreover, such simplification permits a reduction of two variables in the simulation phase.

3.2.2 Topology related documents: Neighbors and Label

Apart JRMs and SoRTBids, the information component contains other two kinds of documents, used for managing the overlay network (Neighbors) and to potentially optimize remote discovery in SoRTGrid (Label). The Neighbors document is provided by PORS (cfr. 2.6) to each FA in order to indicate its neighbors; it is updated periodically or, in case of removal or entering of new Facilitator Agents (like in transient VOs). The Label document is supported but optional. In case of remote discovering, I've previously argued

that forwarding the query through broadcast can be inefficient and that an ad-hoc forward based on consideration of SoRTBid offers of neighbors can be more efficient. The Label document supports this: every Facilitator Agent can build up such document with information regarding the current range of offers for single attributes and broadcasting it to the neighbors, periodically.

```
- <NEIGHBORS>
    <ID_FA>https://notredame.paris.fr/notredame</ID_FA>
    <NEIGH>https://notredame.paris.fr/valdamour</NEIGH>
    <NEIGH>https://notredame.paris.fr/courdemiracles</NEIGH>
  </NEIGHBORS>

- <LABEL>
    <ID_FA>https://notredame.paris.fr/notredame</ID_FA>
  - <RESOURCE>
      <ID_RES>MIPS</ID_RES>
      <MIN_VAL>10</MIN_VAL>
      <MAX_VAL>50</MAX_VAL>
    </RESOURCE>
    <LAST_UPD>2009/04/25 01:20:00 GMT</LAST_UPD>
  </LABEL>
```

Figure 3.4: General schema for Neighbors and Label documents.

A Neighbors document is composed by an <*ID_FA*> that indicates an unique reference on underlying Grid middleware format. It is not a concern how this is indicated from an information component point of view: such information is built by the PORS that belongs to the architecture layer and used by Facilitator Agents. The correctness of such information does not concern the information component, but only that the document is built up properly, following the XML schema. In the same way, for every assigned neighbor a proper <*NEIGH*> tag is defined, containing the id.

A Label document contains the identifier of the Facilitator Agent (<*ID_FA*>). For every attribute in a SoRTBid (<*ID_RES*>), minimum and maximum value (<*MIN_VAL*> <*MAX_VAL*>) offered by current SoRTBid are provided. Moreover, information on the last update of the provided information is supported (<*LAST_UPD*>).

3.3 Implementation of SoRTGrid architecture

The implementation of SoRTGrid architecture is composed by three parts, namely (1) the implementation of the architectural component and their behavior, independently of the underlying middleware; (2) the implementation of a GUI and a tool for building, load and

save SoRTGrid scenarios, and (3) an interface to the Grid middleware (real or simulated) in order to interact with real resources and execution environments.

In this chapter I present the first two points, while the third one is postponed to the next chapter since, as said, SoRTGrid is currently running on a simulated Grid through GridSim; for this, I need to provide an introduction to GridSim and Grid simulation before presenting the implementation of this part.

However, in general the interface to the middleware has to be such that the logical structure implemented at point (1) can be merged on the middleware structure. For the purpose of this thesis, this means that logical structures have to be mapped on proper Grid Services (real or made through logical APIs).

For the aim of this section, I suppose that the architectural components here analyzed work over proper *SRT_GridResources*. Such class is an extension of a Grid Resource (i.e. a Grid Service) that supports time constraints on execution (this assumption is related with one of the four pillars which SoRTGrid design is based on: i.e. assurances on QoS must be provided explicitly and directly by owners that control resources).

3.3.1 SoRTGrid architectural component

The architecture of SoRTGrid has been developed modularly and incrementally, from basic instruments for managing documents belonging to the Information Component up to the global overlay network of Facilitator Agents. The aim of this section is not to present the code but only the peculiar architectural choices made.

First, it is important to understand that such components and communications among them are implemented apart from the underlying Grid scenario and middleware.

This part of SoRTGrid is fully implemented in Java, so different architectural components are implemented as Java classes, while communication among classes is implemented using both local method calls (for local communications among Agents) and Java RMI (for remote communications - i.e. for calling methods of objects belonging to another JVM).

First, I implemented SoRTGrid without the Java RMI support for local simulation. The extension to the Java RMI support is currently a work in progress for allowing to execute a physically distributed SoRTGrid on a real Grid.

3.3.1.1 SoRTBids and JRMs management

SoRTGrid architecture is composed by wrapper classes that manage directly both SoRTBids and JRMs. These classes realize different functionalities for Owners, Users and Fa-

cilitator Agents. More specifically they provide:

- *Building of a SoRTBid/JRM.* SoRTBid class allows to create a SoRTBid starting from SRT_GridResource and the set of parameters needed (e.g. MIPS, percentage, QoS level and so on). It grants the build of a correct XML SoRTBid. In the same way, the JRM class grants a correct definition and publication of a JRM, following the XML schemas presented before.
- *Management of the SoRTBid/JRM.* It allows to update (if possible) the characteristics of a SoRTBid or JRM or its removal and the assignment of a SoRTBid to a given Owner Agent and of a JRM to a User Agent. Moreover, such classes grant the possibility to obtain information on such documents in every state of their life-cycle (e.g. obtaining the assigned Owner Agent for a JRM after the acquisition of a SoRTBid).

3.3.1.2 Owner and User Agents

The Owner Agent class implements mechanisms to create an Owner Agent entity and granting its interactions with SoRTGrid. An Owner Agent creation starts from a set of SRT_GridResources that belong to the Owner Agent. In this context, the Owner Agent is registered, by default, to a proper Facilitator Agent but changes and migrations to other Facilitator Agents are supported for further bids discovery purposes. On the set of the SRT_GridResources, methods for building a given SoRTBid and for supporting automatic definition of a set of SoRTBids are implemented.

Moreover, among the implemented features there is the possibility to provide policies for defining SoRTBids and to have automatic definition of SoRTBids following these policies (e.g. provide only Real-Time QoS SoRTBids, offer only maximum percentage of the resources possessed). Besides, automatic control over SoRTBids submitted to the Facilitator Agents can be performed, in order to avoid situations of overbooking.

As an important remark, the Owner Agent receives such policies only from the physical owner of the resources and, in a running system, the creation of SoRTBids is fully automatic by following the provided policies. Notwithstanding this, there is the possibility to provide manually a set of SoRTBids created by the owner and provided once before the starting of the SoRTGrid framework.

From a behavioral perspective, the Owner Agent implementation contains methods for uploading the SoRTBids on the BidMan service of the Facilitator Agent and for requesting a removal of uploaded SoRTBids from the BidMan.

On the other hand, the implementation of the User Agent is quite different: even if SoRTGrid has a generic approach, I am principally interested in dealing with SRTS, so for now

the User Agent has been implemented with the assumption of the fixed set of jobs (cfr. 2.1.6). This means that the User Agent class allows to create automatically or to insert manually a set of JRMs that represents the set of jobs to execute. JRMs are then managed automatically by the User Agents following, even in this case, policies submitted by the physical user (e.g. period of time when submitting a JRM and so on). The User Agent makes pending (i.e. sends to the Facilitator Agent) one of the JRMs and analyzes the returned SoRTBids, making choices on the threshold value and the characteristics of the JRM. Currently, the possibility to introduce different algorithms for making selection is implemented.

Like Owner Agent, the User Agent class supports a set of operations for controlling and managing JRMs (make them pending, remove them) and for registering the User Agent to a Facilitator Agent.

Both Owner and User Agent classes manage statistics on SoRTBids and JRMs in order to control the trend of its objectives. Moreover, support for implementing artificial intelligence techniques so that they can dynamically adapt their behavior to the results provided by statistics (in order to improve their respective earns). Currently, only a couple of algorithms at the Owner side (K-Means [ARS98] and MeanShift) have been considered, for evaluating adaptation of the Owner Agent in SoRTBids production.

3.3.1.3 Facilitator Agents

SoRTGrid is composed by an indirect graph of connected Facilitator Agents that form the discovery overlay network. The Facilitator Agent class models a logical structure able to provide services to both User and Owner Agents.

First of all, it allows both User and Owner Agents to register and unregister at the Facilitator Agent. For these activities, that are autonomously performed by the agents, routines for controlling the validity of the process are implemented. For instance, when an agent registers, the Facilitator Agent controls if it is not already registered to any other one and, conversely, when an agent unregisters it checks if the unregistration is possible and then performs consistency operations (e.g. remove all active SoRTBids belonging to the unregistering Owner Agent).

Registered Owner and User Agents are allowed to upload/remove SoRTBids and to submit JRMs, respectively. When a JRM is made pending by the User Agent, the local Facilitator Agent performs a time-constrained discovery and selection of SoRTBids in its set of SoRTBids and in those belonging to other Facilitator Agents through the BidMan and the DiPe sub-components.

The BidMan is implemented as an object containing a structure for the Local Repository. The Local Repository is implemented both as a list of SoRTBids and as a file, in order

to maintain a backup copy in case of failure. Operations for the maintenance of the Local Repository structure are implemented for coherency control on the Owner Agents upload/removal of active SoRTBids and automatic remove of expired SoRTBids.

A Facilitator Agent object contains a set of DiPe Services that changes dynamically and permits the interaction with User Agents. A DiPe service is created anytime a User Agent requires to find suitable SoRTBids for one of its JRMs or when a DiPe belonging to an outer Facilitator Agent requires to find SoRTBids for a certain JRM in the Local Repository.

Besides, a Facilitator Agent contains a set of methods that allow it to register to a PORS and getting back neighbors references. Current implementation supports different algorithms for making choices on JRM forwarding in case of Remote Discovery. In particular, broadcast, random and multicast based on neighbors Label document are implemented. In the latter case, two policies are implemented: the *best fitting* that choices to forward the JRM to the neighbors with best fitting attributes and the *cost optimization* one, that forwards to the neighbors with average lower cost of SoRTBids.

3.3.1.4 SoRTGrid checkpointing

Owner, User and Facilitator Agents implementation contains methods for serializing, on a proper XML file, their state in every moment of their life-cycle. This is useful for making checkpoints in SoRTGrid running time, in order to analyze critical situations or to restart the system in case of partial or total failure.

An example of such serialization is provided in Fig. 3.5. The meaning of the tags is trivial; it is important to notice that every aspect of the system is checkpointed.

Excluding tags with trivial meanings or already analyzed before, it is interesting to note that every Facilitator Agent contains the list of associated Owner and User Agents (<*ASSOCIATEDOWNERAGENTS*> and <*ASSOCIATEDUSERAGENTS*>)
with respective SoRTBids and JRMs. Concerning JRM, the Facilitator Agent takes into account the pending JRM and the satisfied JRM in order to restart the system from a given point. The Local Repository state, in the same way, is checkpointed as a set of SoRTBids belonging to Owner Agents. On the contrary, discovery activities are not checkpointed. This is done for a pair of reasons. First, in case of a failure, information on pending JRMs, satisfied JRMs and active SoRTBids sufficies for allowing both parts to easily re-create the previous situation. Moreover, in case of scenarios with many users and owners, the global serialization has to be refreshed instantaneously and much more information has to be saved, so that consistency could be not granted.

The last part of the configuration XML file contains information on the overlay network (<*NEIGHBORHOOD*>) in the form of list of neighbor Facilitator Agents. For every neighbor relationship there is a double direction edge (because the graph is undirected)

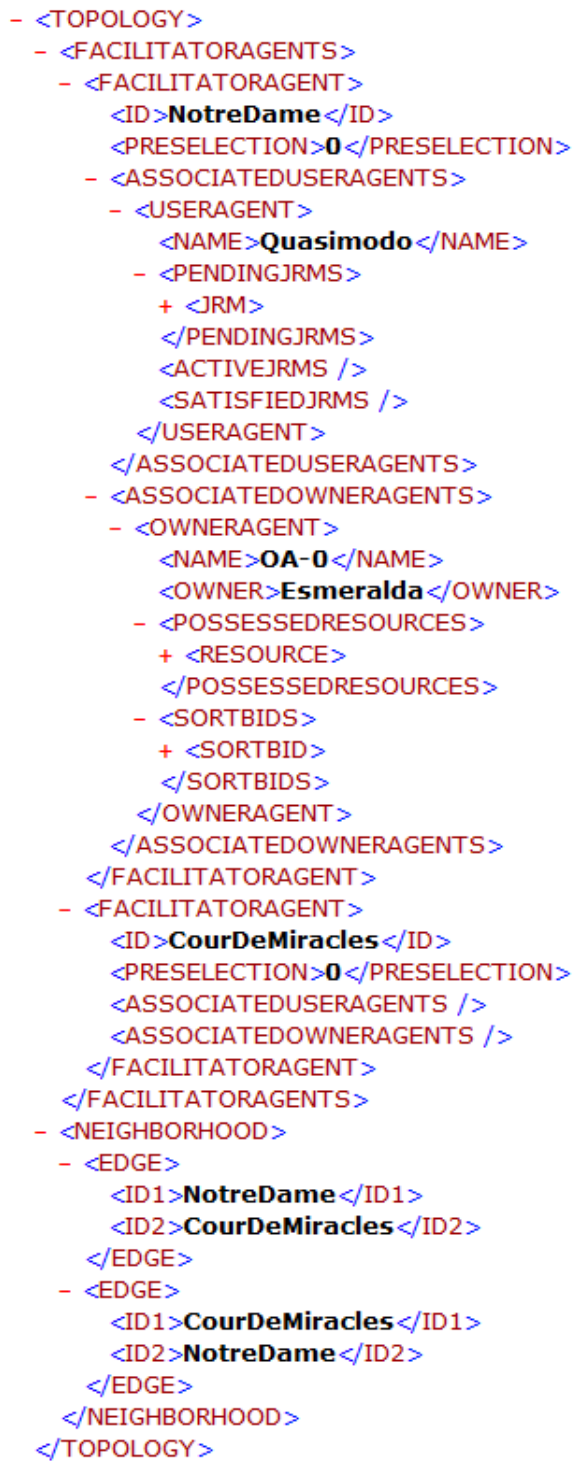

Figure 3.5: An example of a SoRTGrid checkpoint XML.

connecting both neighbors.

3.3.1.5 PORS

The PORS is a proper part of SoRTGrid calculating Delaunay triangulation and defining the neighbor relationship between Facilitator Agents. PORS first aim is to maintain the overlay graph of Facilitator Agents connected, keeping the graph as much balanced as possible. Its behavior depends on the number of Facilitator Agents in SoRTGrid that changes dynamically during SoRTGrid running time, so as Virtual Organizations change and vary in a Grid. For the purpose of this chapter, it is not interesting to present the algorithms, rather to underlie that in SoRTGrid there is only a single instance of PORS, possibly replicated for fault tolerance purpose, especially if the number of Facilitator Agents is high.

PORS is implemented with a proactive approach: when a Facilitator Agent enters or leaves SoRTGrid, re-triangulation is made and new neighbors are notified to all registered Facilitator Agents. PORS supports only two kinds of queries from Facilitator Agents, namely the *registration request* and the *cancellation request* for entering or leaving SoRTGrid. In both cases, the Facilitator Agent is obliged to perform such operation to enter or leave. If this first operation is not made, the Facilitator Agent is not allowed to access to SoRTGrid . In order to recognize Facilitator Agents that leave without sending the *cancellation request*, PORS controls periodically the state of Facilitator Agents and, in case of missing answer, it remove the Facilitator Agents from SoRTGrid and re-triangulate.

As a final remark on PORS, in my opinion it is not a strong assumption to assume a single PORS because, even if the Grid can be wide, Facilitator Agents act as sort of super-peers, so their number is small in comparison to the total number of Owner and User Agents.

3.3.2 SoRTGrid GUI

On a real Grid, SoRTGrid is dynamically defined by entering and exiting of autonomous Facilitator, Owner and User Agents in the framework. In this sense, SoRTGrid structure and organization on a Grid depend strictly on choices made by such entities.

From a SoRTGrid perspective, this makes the framework totally autonomous and potentially unaware regarding the number and nature of the entities that, in a given moment, belong to it. From this, it is simply to conclude that a mechanism for building a SoRTGrid scenario and having a global visualization of the state of SoRTGrid in a given moment could be useful but difficulty to obtain, especially in a centralized way.

On the other hand, it is important to simulate different scenarios and make tests in order

to debug different aspects of an implemented distribution of SoRTGrid, in order to be sure that the framework works effectively under the specification presented in the previous chapter.

Because the complexity of building manually scenarios for SoRTGrid (through definition of the previously analyzed XML files) is huge (all Agents, the overlay network, JRMs and SoRTBids have to be defined and configured), I implemented a proper tool for supporting a fast and simple configuration of SoRTGrid through a graphical interface and an autonomic mechanism for random generation of some parts of SoRTGrid scenarios.

SoRTGrid interface is composed by three parts that allow to build up SoRTGrid scenarios under different perspective, namely the user, the owner, the framework one.

3.3.2.1 User side definition

The user side part of SoRTGrid interface (Fig. 3.6) allows to define the set of User Agents and JRMs and to assign JRMs to User Agents and make JRMs pending on SoRTGrid. In this first part, User Agents are still not connected to any Facilitator Agent. This part of the interface can be used in two ways: 1) statefully, to build up the scenario and then run the simulation, 2) to create or destroy User Agents at runtime, during SoRTGrid execution.

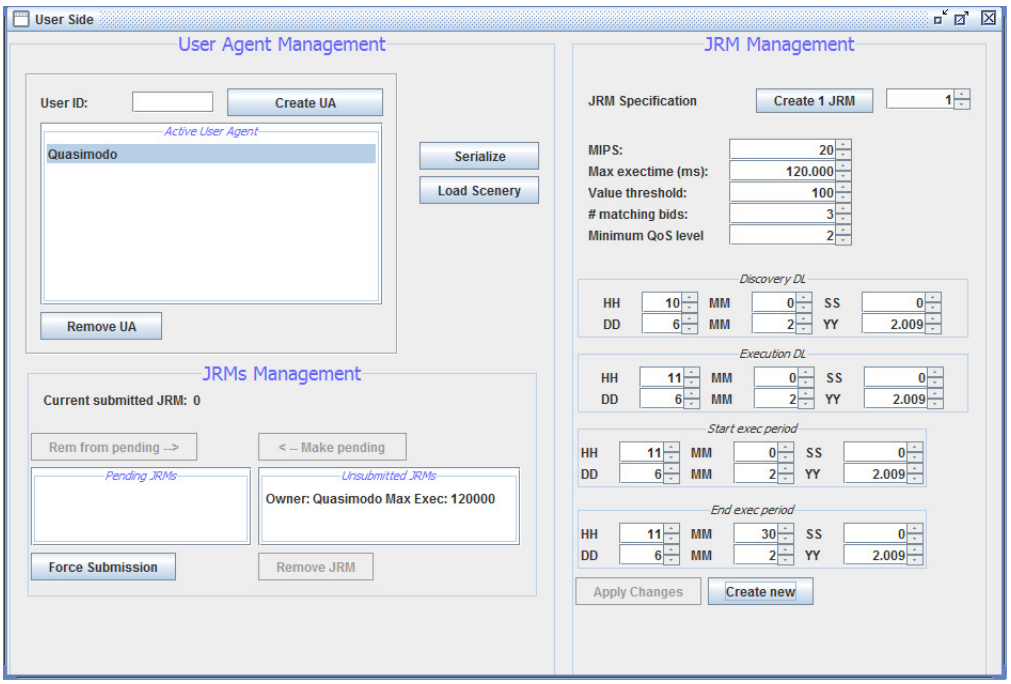

Figure 3.6: User Agent and JRM management.

The created JRMs and User Agents can be serialized on a configuration XML file and checkpointed at runtime automatically. Moreover, sets of JRMs can be randomly generated starting from a subset of values provided in the JRM Management mask.

On an effective Grid, the creation of a User Agent is automatically done when a user with jobs needs to join SoRTGrid, and JRMs are defined automatically starting from jobs time constraints and user policies.

3.3.2.2 Owner side definition

In the same way, Owner Agents and SoRTBids creation is supported in this frame (Fig. 3.7). First of all, sets of logical resources are created, and simultaneously, a set of Owner Agents is built. Through this interface both sets can be managed randomly, starting, at resource side, from the type of architecture and average computational power. Owner Agents are barely a name, an unique id identifying universally the Owner Agent in SoRT-Grid. Random assignment for coupling Owner Agents and resources is supported. At present there are no policies for balancing the distribution of resources on Owner Agents and vice-versa.

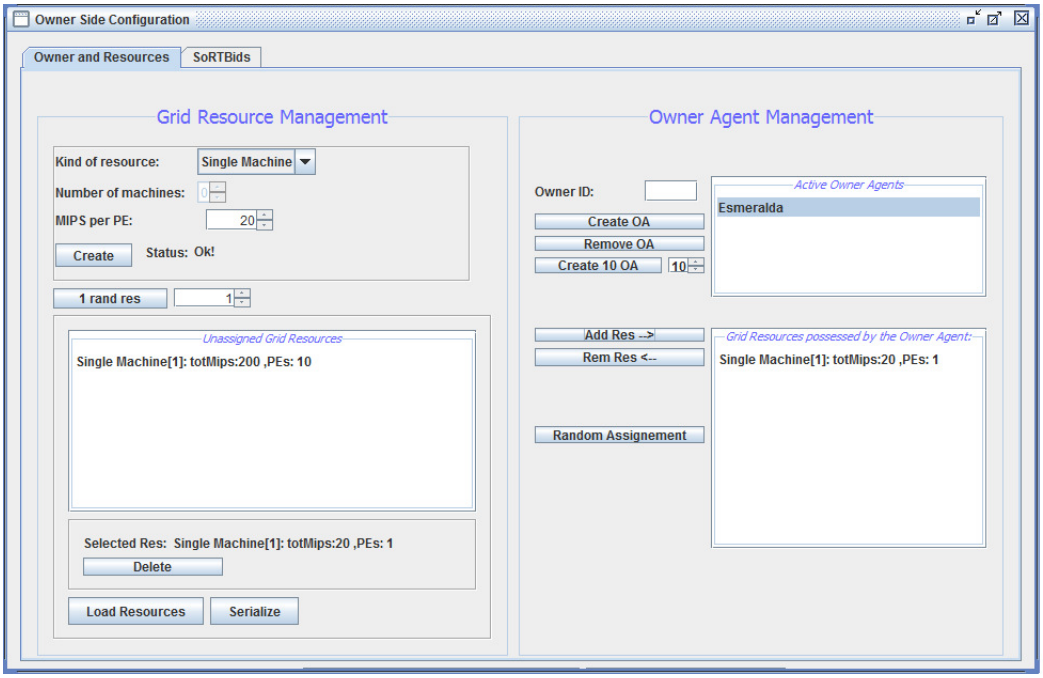

Figure 3.7: Resource and Owner Agent management.

On a real working SoRTGrid, the underlying resources are physical, provided and defined

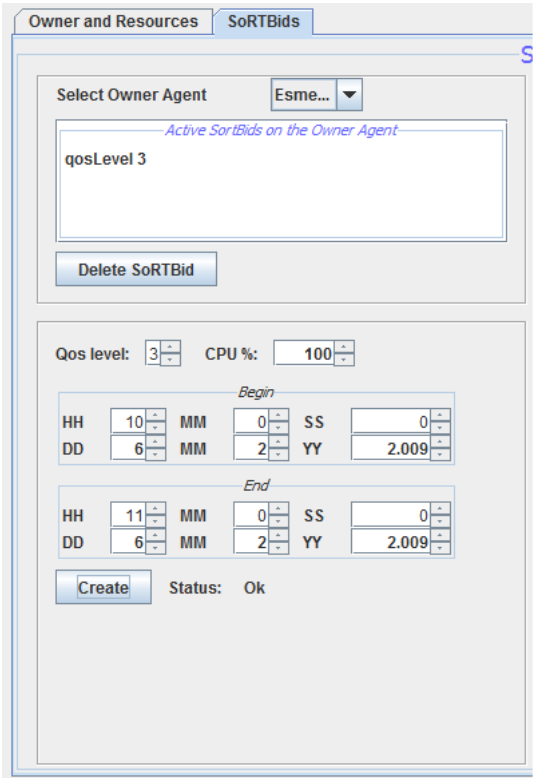

Figure 3.8: SoRTBid building.

explicitly by the Grid owner who has proper responsibility on correctness of description of provided resources.

For every Owner Agent, the creation of SoRTBid is supported (Fig. 3.8). This building is fully manual or totally random with trivial controls on overbooking in SoRTBid production. In a real context, this is demanded to the Owner Agent through policies provided by Grid owner.

3.3.2.3 Facilitator Agents and overlay network

Topology frame is the core component of the interface and requires that both owner and user sides have been previously defined. In Topology section Facilitator Agents are created. Like Owner Agents and User Agents, they are barely a unique id. After creating a set of Facilitator Agents, it is possible to manually or randomly assign Owner Agents and User Agents to Facilitator Agents.

Moreover, it is supported the manual definition of the overlay network among Facilitator Agents, by defining single neighbor relationships on any created Facilitator Agent.

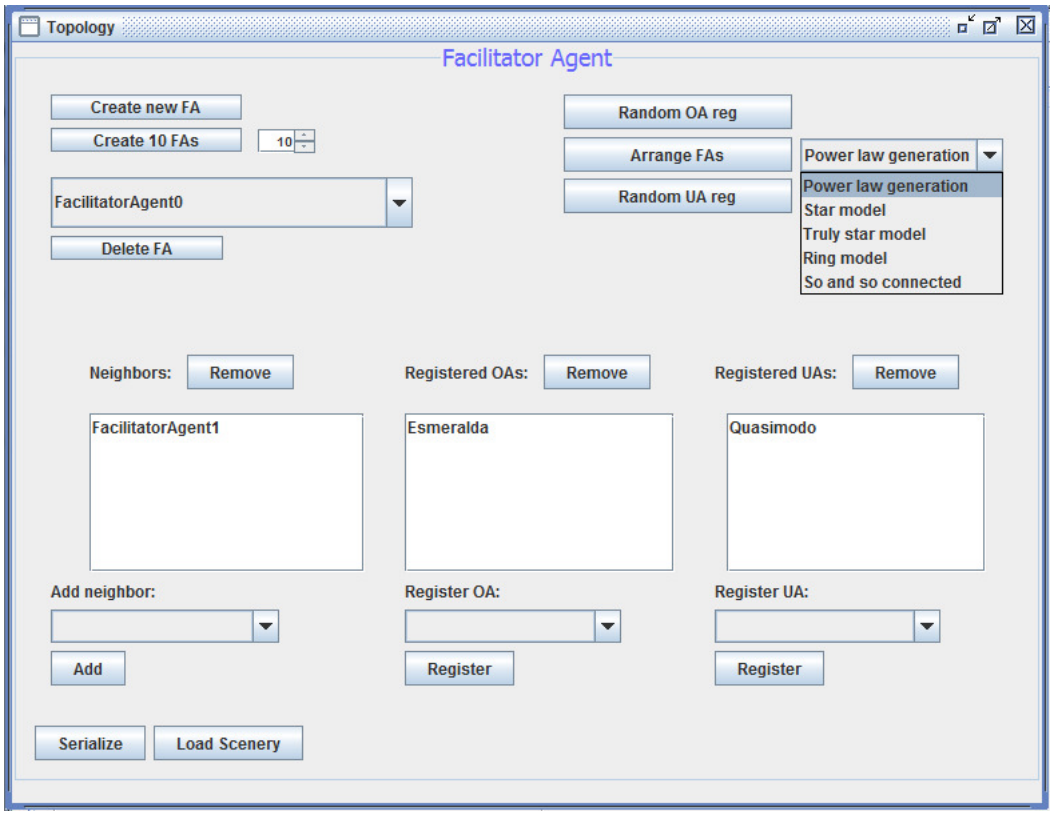

Figure 3.9: Facilitator Agents and topology definition.

The operation of creating manually the topology can be a good choice when dealing with few Facilitator Agents. As remarked and in general, the topology is created and maintained automatically by a proper application (the PORS). The interface simulates the PORS behavior by granting an automatic arrangement of the set of Facilitator Agents through the use of different algorithms (Power Law generation, Star Model, Truly star model, Ring model), plus the standard algorithm that remains the Delaunay Triangulation.

In a real context, Facilitator Agent enters SoRTGrid without any Owner/User Agent assigned. Owners and Users are associated with the Facilitator Agent when they are created or migrate.

As a final remark, I present here (Fig. 3.10) an example of a simple SoRTGrid scenario fully generated through the interface. The whole scenario is fully serializable in XML files, like those presented in the previous section. For this, different scenarios can be created and saved for further modification, both before and during a SoRTGrid execution.

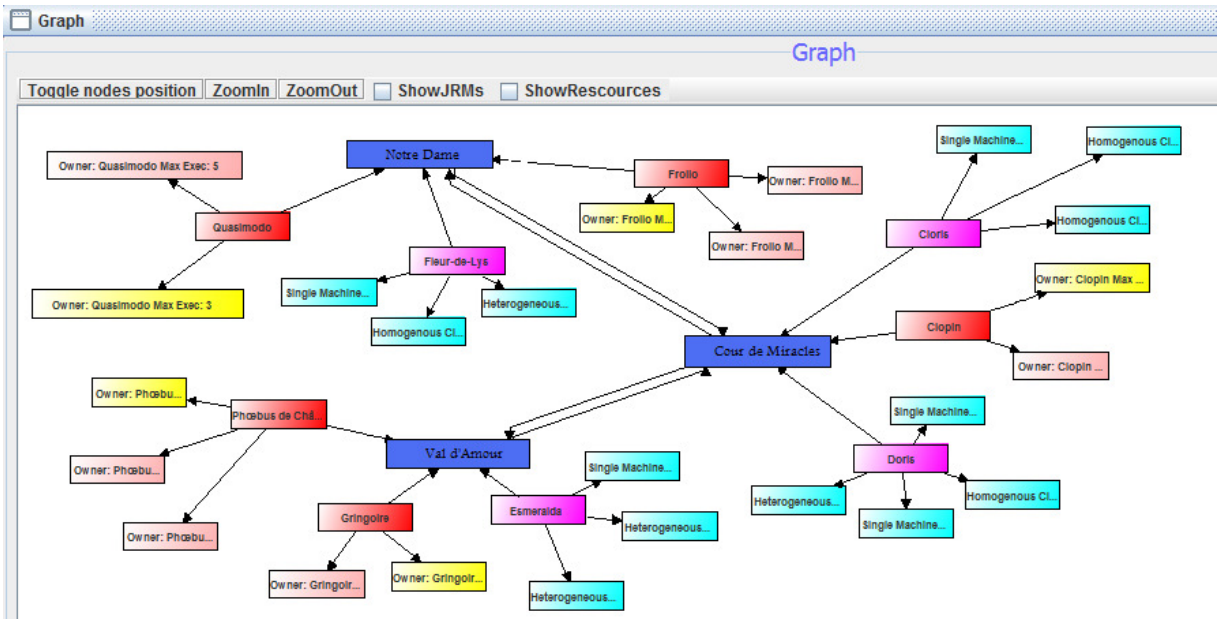

Figure 3.10: Generated scenario in SoRTGrid. *Blue* rectangle represents Facilitator Agents; *purple* and *red* rectangles represent Owner and User Agents respectively; *Light blue* rectangles represent physical resources; *pink* rectangles stand for JRM while *yellow* ones represent the current pending JRM for the given User Agent.

3.3.3 Porting the logical structure on a real Service Oriented Grid

The presented framework can be differently ported to a Grid scenario that can be real or simulated. The current work in progress is the porting of SoRTGrid to GridSim as a Grid logical simulator. This step, that is not still finished, will be presented in detail in the following chapter. However, because a real Grid is generally Service-oriented, for the sake of completeness I present some guidelines for porting the current SoRTGrid implementation on a Service-oriented middleware, using the structure of GT4 as example. Under this perspective, this work is a further step to carry out after the porting on GridSim is completed and tested with satisfying results.

Trivially, porting SoRTGrid to a Grid middleware means that the fundamental logical structures need to be implemented as structures recognized by the Grid middleware. Because I'm dealing specifically with Service Oriented Grid, this means that such structures have to be implemented as Grid Services.

Moreover, the porting must be such that, during the execution of SoRTGrid, any possible dynamic behavior and operation must be supported (e.g. entering or leaving of any possible agent) as an interaction with middleware structure and algorithm. This is an important point in a Grid perspective, so a further explanation is required. For instance, suppose that a new Facilitator Agent need to join SoRTGrid. In SoRTGrid design and algorithms presented before, this is possible contacting the PORS. If PORS is an external application, the middleware is not involved and this is not "Grid". Conversely, if PORS is a Grid Service and the registration to SoRTGrid passes through PORS as a Grid Service and involves the middleware mechanisms for interacting with Grid Services this is "Grid".

This means that SoRTGrid works effectively over Grid middleware through the implemented structure described before, but must interface and work together with middleware (this is the aim of the SoRTGrid interface with the middleware - cfr. 3.3).

For this, I present here which operations (cfr. Fig. 3.11) must be followed in defining an interface of SoRTGrid with Globus Toolkit 4, in order to define the basic steps, but without investigating implementation details.

- *Implementing PORS and Facilitator Agents as Grid Services.* The PORS and the Facilitator Agent have to be implemented as Grid Services in order to permit the building of an effective working SoRTGrid. The building of a Grid Service from a proper definition of a Java class is supported by proper tools like Grid Development Tools (GDT) [gdt] for Eclipse [ecl].
- *Registering PORS-GS to an Index Service (MDS).* The publication of PORS as a Grid Service (PORS-GS) is the first fundamental step because new coming Facilitator

Agents, that physically build up SoRTGrid, need to know the PORS Grid Service address. For this, an instance of a PORS-GS needs to be registered to a GT4 Index Service (MDS) in order to be discovered by organizations belonging to the Grid that aim to join SoRTGrid with their Facilitator Agents.

- *Registering the Facilitator Agent to PORS and MDS.* A new coming Facilitator Agent Grid Service (FA-GS) needs to register first to PORS to receive its neighbors and, afterwards, it has to register to MDS in order to be public and discoverable by Grid users.

- *Building user and owner sides.* SoRTGrid stakeholders need to share resources under SoRTGrid conditions and standards (owners) and to use SoRTGrid services in order to find resources for time-constrained jobs (users). The participation of such Grid owners and users in SoRTGrid is subject to the creation and assignment of a proper Agent (Owner or User). Such Agent operates in the physical user/owner stead in SoRTGrid, receiving resources to share and policies (owner side), or jobs to execute and policies (user side). To obtain this, a user/owner registers to a FA providing those information; the FA generates a proper Agent in SoRTGrid, building back an Agent Control Grid Service (AC-GS) at middleware layer. The Grid real user/owner (identified and universally recognized on the GT4 Grid) is granted with exclusive access to it. The user adopts the AC-GS to interact to the SoRTGrid Agent for providing new policies, adding new resources to share and jobs to execute. An AC-GS is tied up with a given FA-GS.

- *Removal of SoRTGrid components.* During SoRTGrid execution, removal of components (if correct and possible) is represented by elimination of proper Grid Services and de-registration from the MDS. For instance, if a user/owner aims to leave SoRTGrid, its Agent in SoRTGrid is removed and the relative AC-GS is deleted. In the same way, if an FA-GS leaves SoRTGrid, the connected AG-GS is migrated to another FA-GS through proper modifications of single AC-GS configuration.

At present, this process of porting SoRTGrid on Globus Toolkit 4 is being realized. I have still to implement some Grid Service instances of the SoRTGrid logical structures and to test the working system. Regarding this point, a last remark concerns the Grid based on GT4. Because an interesting testbed requires many physical machines with GT4, it is an aim of this phase to use many Virtual Machines in order to dispose of a large enough Grid. On every Virtual Machine GT4 will be installed, constituting a projection of a full working machine.

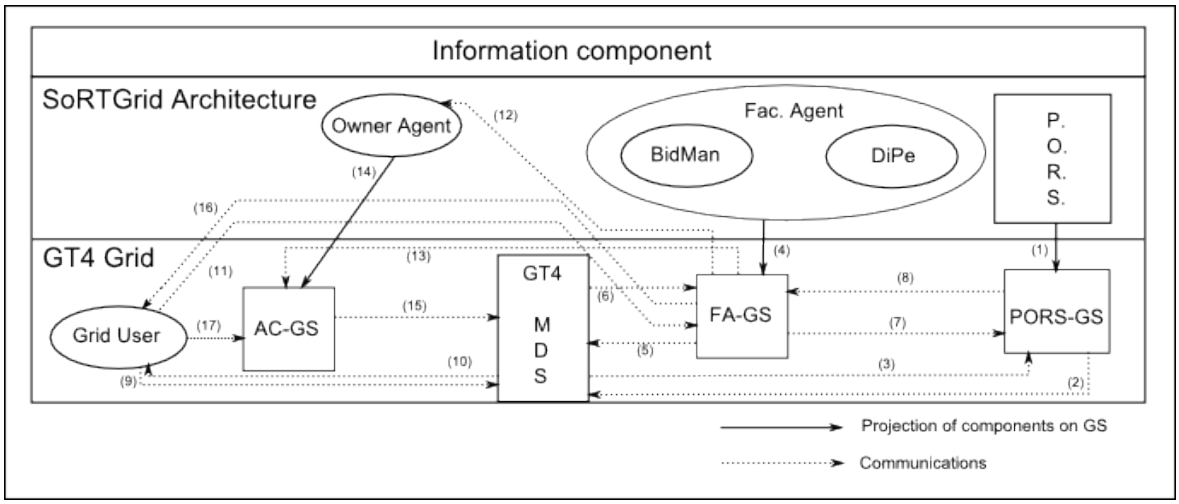

Figure 3.11: *Example of the implementation of SoRTGrid on GT4. (1) PORS is projected to PORS-GS, (2) PORS-GS registers to MDS and (3) receives acknowledgment. (4) FA is projected to a FA-GS which first (5) queries the MDS to (6) obtain the PORS-GS URI. (7) The FA-GS queries the PORS-GS and (8) obtains neighbors. (9) A Grid owner/user requires to the MDS and (10) receives the address of the FA-GS. (11) It contacts the FA-GS which first (12) creates a proper Owner/User Agent in SoRTGrid, then (13) creates a proper AC-GS where (14) the Owner/User Agent projects to. (15) The AC-GS is registered to the MDS, (16) the confirmation of entrance in SoRTGrid and the URI of the AC-GS are sent back by the FA-GS to the user, who can then (17) access the AC-GS.*

Chapter 4

Experimental evaluation of SoRTGrid overlay architecture and prototype testing

When proposing new solutions in computer science, in general different steps have to be followed[CGM+89]. First, the definition starts from a theory or an idea (Theory), basically from the analysis of the state of art an unsolved problem arises. Second, a proposal for solving the problem is defined (Modeling), and then an applicative solution for the given proposal is provided (Design) and, if necessary, implemented (Prototyping).

Regarding SoRTGrid, the development of such framework has covered all the four steps: in Chapter 1, I analyzed the open problem of managing time-constraints on Grid, while in Chapter 2 SoRTGrid overlay is provided, as an architectural solution to the problem; finally, Chapter 3 presented the prototype implementation on the designed architecture.

At this points, two issues arise. First, the overlay architecture proposed in SoRTGrid (User and Owner Agents, overlay network of Facilitator Agents, presented in Chapt. 2) constitutes a new way to approach the resource discovery and acquisition in a Grid, surely very different from the typical procedure adopted by current middleware. In fact, in a middleware the owner publishes information regarding its resources to a proper service (e.g. Index Service); the user queries the same entity to retrieve a list of resources (often independently of its requirements). Such service independently manages the discovery activity and the upgrade of the resources information. In this context, both owners and users have a passive role. In the SoRTGrid approach owners and users become active and the middleware acts under their requirements and requests in a totally opposite way. For this, it is fundamental to investigate if the proposed architecture is suitable for the management of jobs with time constraints over a very heterogeneous context referable to

a Grid. I call this issue as *SoRTGrid overlay evaluation*.

On the other hand, the second issue regards the testing of the current SoRTGrid prototype (presented in the Chapt. 3) over a first working Grid (i.e. the logical Grid simulated through GridSim), in order to verify both efficiency and correctness of the implementation. I call this problem as *SoRTGrid prototype testing*.

Both objectives have to be realized through experiments, by defining proper testbeds and executing tests aimed at evaluating the choices made in both phases. However, the experimental approach is different, because aims are opposite. In fact, the evaluation of the overlay aims at verifying the correctness of the overlay solely, so it requires a qualitative approach (i.e. well-defined and minimal set of tests at a high abstraction level and on a reduced set of variables); on the other hand, the testing of a prototype aims at evaluating its implementation through a quantitative approach (i.e. on a huge set of scenarios and on every variable regarding any implementation aspect).

Therefore, it is not possible to convey both experiments in a single one, (i.e. evaluating the overlay by running tests on the SoRTGrid implementation), so two different experimental phases are required.

To this regard, this chapter is dedicated to the SoRTGrid overlay evaluation experiments and to the porting of the current SoRTGrid prototype on an effective logical Grid. In particular, the main aim is the definition of a strategy for an experimental evaluation of the SoRTGrid architecture for SRTSs. Regarding the prototype, this chapter contains an analysis of the current available tools for building a logical Grid on which to port the prototype, and a final summary providing details on the way to port SoRTGrid implementation over the selected tool (GridSim). However, the experimental test of the ported SoRTGrid implementation is not a concern of this thesis.

The chapter is organized as follows: the next section defines some motivated starting points for defining a good experimental context for SoRTGrid evaluation and the related issues. Hence, Section 4.2 investigates the best technique to test the overlay architecture under the experimental requirements analyzed in Section 4.1 and fixes the assumptions for the experiments on the selected technique; moreover, a strategy for evaluating the overlay is provided. Section 4.3 is dedicated to the prototype testing, presenting both the Grid tools analysis and details on the porting to one of them (GridSim).

4.1 An experimental context for the overlay evaluation

SoRTGrid overlay architecture is very generic and depends on many variables (QoS level, deadline and activation interval, period of resource availability and so on). This allows SoRTGrid to operate potentially on any kind of application. For this, an exhaustive architectural evaluation on every possible applicative scenario is impossible to realize. Besides, a SRTS has a high QoS demand (in term of resource availability) and is made by jobs with deadline and unpredictable activation, constituting an application scenario with very strict constraints. For this, the idea is that if SoRTGrid provides satisfying results for such application scenario, than it can be assumed that it is suitable even for managing other applications with less QoS demand. Moreover, if only SRTS are considered, the number of involved variables is sensibly reduced, as can be easily noticed, for instance, by comparing XML files in Fig. 3.2 and Fig. 3.3. For this reason, *I limit the evaluation of SoRTGrid architecture to SRTSs.*

The evaluation based on SRTS solely reduces the complexity of the experiments at application side, but still too many variables are involved both on the current framework implementation and in an effective Grid (real or simulated). Even in these cases, it important to make strong hypotheses but it is quite impossible because the current SoRTGrid framework and Grid scenarios (logical or real) are based on many variables that are hard to reduce. To this regard, in this section I point out which characteristics of both SoRTGrid implementation and Grid environment make difficult the overlay evaluation.

Regarding the current SorTGrid implementation, unwanted complications are four:

- *Complex components.* The overlay architecture is implemented over different standards and layers (e.g. the document layer made of XML and configuration files) and uses components that are fundamental for the execution of the framework on an effective Grid (e.g. the PORS component). Although, for evaluating the efficiency of the overlay architecture these parts introduce unpredictability and unwanted delays, it is not possible to execute the framework without them.

- *Continuous time.* SoRTGrid operates, trivially, on continuous time. While evaluating the overlay, it is important to have a precise control over time, in order to have a simpler evaluation of causes of delays or degradation. In this sense, it is not possible to execute the current SoRTGrid implementation on a discretized time.

- *Latencies and variable length of an operation.* In a context complicated by continuous timing and complex components, two issues arise. First, latencies not related with the overlay architecture can be detected and it can be difficult to precisely define

their amount and cause. Secondly, the same operation is difficulty upper-boundable in its length and it can have different durations.

- *Required porting on a SOA middleware.* The framework can effectively run only if the components of the overlay architecture are mapped on proper middleware structures[1], so it is not possible to run the framework without porting it over a Grid.

The previous points underline that it is impossible to make assumptions on a running SoRTGrid framework. In particular, the last one underlines *the need of an effective porting on a Grid in order to be executed*, so it is important to point out issues related on a Grid platform, from an evaluation perspective. To this purpose, the use of an effective Grid (both real or logical) has important drawbacks I analyze briefly in the following.

4.1.1 SoRTGrid experiments on a real Grid

The evaluation of an overlay architecture on an real Grid is complicated by four limitations that make the real context unsuitable for complete experiments:

- *Real Grids have a little availability.* Grids need to be large for testing a framework like SoRTGrid; in fact, there is the need of enough variety in scenarios and of satisfying results in the majority of the cases, in order to prove the framework architecture operates correctly. To this purpose, current real Grids are not big enough or not accessible enough, so the access to a large set of resources with a controlled behavior (e.g. QoS level granted) is hard to assume.

- *Real Grids are too heterogeneous.* Real Grids are general purpose and support potentially any kind of resource, from physical to logical, from hardware to software. Moreover, they are distributed through different POs, so communication latencies for information exchange are non deterministic. Finally, communication success is not granted, because real Grids cannot be assumed as fault-free. Because Grid frameworks are *specialized* only on particular services (cfr. the QoS management layer in Chapt. 1), then there must be the possibility to consider only characteristics of the Grid concerning the services implemented by the framework. For this, it is always necessary to make particular assumptions on those parts of the Grid that are not directly related to the framework focus. Such assumptions have the aim to reduce the complexity of the scenario and the number of variables considered, so that an evaluation of the test results can be simpler. For instance, while testing the behavior of a set of algorithms of the framework, it could be useful to suppose to work on

[1] As explained in 3.3.3.

fault-free resources so that failures or bottlenecks can be supposed to regard only the algorithms themselves, and are not caused by failure of the involved resources.

- *Complex implementation.* Real Service Oriented Grids require the installation and configuration of the middleware, registration of Grid Services to Index Services, explicit management of security issues and so on. This comes to a significant complication in building Grid scenarios with only simulation purposes. Moreover, it is not possible to assume a full correctness on the behavior of the configured architecture. On the contrary, a simulated Grid reduces the complexity and the time required for building up an effective working scenario, avoiding the need to manage useless aspects of Grid for the simulation purpose.

- *Hard assumptions.* As said in the previous sections, it is important to made assumptions on a scenario, in order to concentrate only on core aspects and, then, simplify the readability of the simulation results. A real Grid is a working architecture on which it is quite impossible to make assumptions, because of its non deterministic behavior: for this, any assumption on a given behavior can be wrong. Moreover, in case the testbed results are tied up with respect to those assumptions, a scenario that effectively behaves differently from the expected can invalidate results and the demonstration of the thesis.

These considerations underline the unsuitability of a real Grid for testing the SoRTGrid overlay model. Regarding the use of a simulated Grid, these issues can be partially solved by properly tuning a Grid simulator. However, the use of a toolkit or, in general, a black box software that simulates a Grid, does not allow to have a deep control on the behavior and correctness of its routines and procedures; for this and because at this point the aim is to evaluate an overlay for managing strict time constraints, the use of a *high level abstraction of a Grid as scenario for evaluating the SoRTGrid overlay* is a better and simpler choice.

From the previous analysis, it is clear that is impossible to use both an effective Grid and the current framework implementation. This takes necessarily to manage three issues in order to evaluate SoRTGrid: (1) Regarding the underlying Grid issue, there is the need to provide a proper abstraction of the Grid, avoiding the use of an effective one, in order to have full control on the behavior, with the possibility to make assumptions that can be assured as valid during the experiments. This aspect is related with the technique used for the experiments (simulation, emulation, in-situ), because any technique regards a possible abstraction for the resource scenario on which the experiment is made. So the evaluation of the most suitable technique for SoRTGrid overlay requirements is needed. (2) On the other hand, there is the need to define both the assumptions and an experimental strategy in order to execute experiments for evaluating the overlay. (3) Finally, because the use of

the current implementation of the framework is unsuitable as a platform for experiments, there is the need to select another tool on which to map the SoRTGrid overlay.

4.2 Definition of an evaluation technique and experimental assumptions

Since the experimental aims for the SoRTGrid overlay evaluation and the prototype testing are different, even the evaluation techniques can differ. In general, an evaluation technique has to be *reproducible* (it has to be such that other researchers, when executing the same experiments can obtain the same results), *extensible* (there can be the possibility to extend the scenario or the algorithms tested in order to make a comparison with previous results) *applicable* (experiments have to be based on realistic hypotheses and variables), *revisable* (if hypotheses are not verified there must be the way to correct and improve them). Regarding the experiments on a real Grid analyzed previously, another drawback is that real experiments lack in significance when dealing with evaluation aims, because are generally not reproducible, and extensibility, applicability and reviseability are hard to obtain.

Regarding the evaluation of the SoRTGrid overlay, the experimental technique has to provide the possibility to make high level and strong assumption on the Grid scenario. As stated in [GJQ09] there are three main methodologies:

- *Simulation.* A computer simulation is an application that attempts to simulate an abstract model of a particular system. The abstraction level is very high and results are extremely reproducible. In general, simulation attempts to test an architecture in conditions otherwise difficult to obtain on a real system. Regarding distributed systems, there are different simulators like NS-2 or GTNets. Moreover, SimGrid [sim] and GridSim [gri] are successful simulators for Grids.

- *Emulation.* A computer emulation allows to test the *real application or a prototype* on a precise reproduction of the real resource set. In comparison with simulation, the resource scenario is effective; as instance, Grid simulators are based on representation of Grid structures but the middleware behavior is not defined. A Grid emulator implements the full middleware layer and not only the set of resources. For this, the emulation is considered less abstract than simulation. Examples of hardware emulators are GXEmul [gxe] that emulates different early PC architectures, and MAME [mam], the very famous emulator for coin-op videogame hardware. A Grid emulator is MicroGRID [XDCC04].

- *In-situ.* In situ experiments offer the greater realism because real applications run on real controlled hardware. This does not mean to run on a real scenario for the

application but on a scenario realized on properly allocated physical resources. There are many distributed clusters that are dedicated to support in situ experiments like PlanetLab [pla]. From a Grid perspective, the Grid'5000 [CCD+05] supports in situ experiments.

The evaluation of the previous methodologies takes me to consider **simulation for evaluating the SoRTGrid overlay** since it allows to use an abstraction of the Grid at high level without the need to use a working reproduction of the Grid resource scenario. The other two techniques (emulation, in-situ) are instead suitable for *testing the SoRTGrid prototype*, where an effectively working resource scenario is needed for the prototype execution, as explained before. At this point and in order to make exhaustive simulations sufficient for an evaluation of the overlay architecture, proper assumptions are made.

4.2.1 SoRTGrid evaluation through simulation

Notwithstanding the reduction of the SoRTGrid overlay evaluation to SRTS only, the SoRTGrid overlay has still a high number of variables because no limitations in the possible configurations of the set of entities are defined (i.e. the number and the possible behavior of User, Owner and Facilitator Agents). In fact, SoRTGrid overlay is fully distributed and can span over Grids of any dimensions. In the same way, the relations between entities (e.g. how Owner and User Agents are distributed over the set of Facilitator Agents) can not be foreseen, and entities, because are autonomous, can have any behavior or policy, entering or leaving SoRTGrid as they need.

All that introduces variables that are important in testing the efficiency of an effective systems but lacks in significance when dealing with the architectural design solely, corresponding, even in this case, to useless complications. For this, it is necessary to define some bounds for the Grid side (as explained before), the overlay network characterization and the simulation behavior. In particular, I make the following assumptions:

GRID ASSUMPTIONS:

- *Abstract representation of the Grid.* The complexity of the Grid scenario is represented at a high level of abstraction by defining proper configurations of the set of bids and JRMs. The variability of the Grid is defined by the variation of the set of active bids and JRMs during the simulation.
- *Computational resources only.* The evaluation based on SRTS takes to consider only the computational resource as a single quantitative attribute.
- *Fixed set of resources.* I consider only a fixed set of computational resource. In any moment of the simulation all or only a part of the resources are active, but the

whole set does not change (i.e. no new resources can join nor definitively leave the simulation). The aim of such point is to reduce the non determinism.

- *No latencies.* Latencies are not considered in communication between entities. Even if this is an important aspect when dealing with Grid, it does not concern the evaluation of the architecture. On the contrary, it is tied up with the test of the implementation.

- *Fault tolerance.* All entities are supposed to be fault tolerant, so I do not consider failures of entities involved. Under this assumption, unexpected situations in the simulation can be related only to the overlay behavior and not to problems concerning the Grid scenario.

OVERLAY ASSUMPTIONS:

- *Deterministic and fixed set of behaviors.* Any Owner and User Agent can have only a given policy taken in a set of fixed ones and make always the same choice when a set of conditions is verified. The aim of this assumption is to reduce the complexity given by the total freedom in defining artificial intelligences for Owner and User Agents, like expected in a real context.

- *Fixed set of jobs.* Like resources, I consider only a fixed set of possible jobs in every simulation. This means that there are no entries (or exits) of new users and the non determinism of Grid users is reduced.

GLOBAL SIMULATION ASSUMPTIONS:

- *Discrete timing.* I assume to work with a discrete time so that any operation can be made in a fixed amount of instants and so that in any instant a given set of operations can be made as atomic. This eliminates the difference in making the same operation by two instances.

- *Centralization.* The overlay is tested using a single central entity that controls all the operations from the resource and job side. This entity can simulate a distributed SoRTGrid (with different Facilitator Agents) but there must be a centralized way for synchronizing all entities. In this way we have an unique management and control of the operations during the simulation.

The defined assumptions simplify the number of variables, improving the analysis and readability of results. In particular, the previous assumptions define a maximum set of variables involved in the simulations. It is trivial to understand that *not any possible combination of values for these variables is significant.* For this, the next step regards

the definition of proper simulation testbeds by selecting only some combinations (in the following, *configurations*) of variable values that are significant for a complete overlay evaluation. Both the precise definition of the set of variables and the testbed definition will be analyzed deeply in the following chapters.

4.2.2 Simulation tools and experimental strategy

A last issue arises before defining the configurations and related testbeds and it regards the selection of a simulator. In particular, it is important to check out if *exists a Grid simulator able to support the previous assumptions*, in order to understand if it can be used for executing the testbeds or if the implementation of a proper one is a better choice. After selecting a simulation tool, a proper strategy for making tests can be defined.

To this purpose, I made an evaluation of different Grid simulators both for the overlay evaluation and the prototyping test. Regarding the first case, the aim has been to find a tool for simulating the overlay only on a abstraction of a Grid; in the second case, the focus has been on finding the best tool for simulating a Grid on which to port the prototype for testing purposes.

The tools considered and their evaluation are presented in the following section. However, from an overlay evaluation perspective, the result has been unsatisfying. In particular, all Grid simulators model an effective Grid and, in general, a distributed system, so the required level of abstraction for the resource scenario (based on the representation of the Grid through the set of bids) is impossible to obtain. For this reason, and because *the assumption on experimenting over an abstracted representation of a Grid is fundamental* in order to reduce the number of variables, my choice has been to implement **a proper simulator, SoRTSim, centralized and based on threads to simulate the behavior of the overlay of SoRTGrid**.

The implementation of SoRTSim solves the simulator problem. At this point, it is important to define an *experimental strategy* that allows to evaluate all features of the overlay architecture under the assumptions defined before. In particular, because the overlay proposed has never been evaluated on any distributed scenario, the idea is to execute simulations on **different user and resource scenarios** (by properly defining different *configurations* of the set of bids and JRMs), from simple and static to complex and dynamic (similar to the behavior of a Grid). The aim is to use configurations that differ by few characteristics at Grid resource side (Owner side), at job side (User side) and at overlay side (Facilitator Agents side) so that the impact of the single modification on the performance of the architecture can be evaluated. In fact, the direct simulation of all features of SoRTGrid on a complex Grid provides results that can be difficult to relate to the single variables involved, so a gradual approach is preferred.

Both these two aspects, together with the evaluation of the results of the tests will be analyzed extensively in the last two chapters. More specifically, the specification of SoRT-Sim, the mapping of SRTS and the definition of the simulation variables are presented on Chapter 6; the configurations, the testbeds definition and the analysis of the simulation results are shown in Chapter 7.

4.3 Porting the SoRTGrid prototype on a logical Grid

In this section we analyze what is a suitable simulator to test the SoRTGrid prototype on a logical Grid and how to perform the porting to such simulator. The choice of a Grid simulator depends on the features it supports and on the characteristics of the application to simulate, in this case the current SoRTGrid prototype. Regarding the first point, I considered a "good" logical Grid simulator the one able to support services for simulating big grids easily, allowing both a *simple and quick definition* of Grid scenarios and offering many *possibilities of customizing* the Grid behavior. The characteristics of the application are likewise important as they take to the selection of a Grid simulator that allows an easy and quick porting of the application to it, allowing correct and controlled simulations.

Currently, there is a plenty number of simulators. Because a full analysis of them is quite impossible, I considered surveys and comparisons in literature between the most interesting proposals [SCV+08] [GJQ09], and, as the result of such analysis, I have selected GridSim [BMA02] as a possible simulator for SoRTGrid requisites.

An exhaustive introduction to GridSim is really difficult and long at this point, because the main aim is to understand in which way it can be used for supporting a test of the prototype. For this, I try to point out only the features that make GridSim a "good" Grid simulator for SoRTGrid.

Functionalities	GridSim	OptorSim	Monarc	ChicSim	SimGrid	MicroGrid
Data replication	Yes	Yes	Yes	Yes	No	No
Disk I/O overheads	Yes	No	Yes	No	No	Yes
Complex file filtering or data query	Yes	No	No	No	No	No
Scheduling user jobs	Yes	No	Yes	Yes	Yes	Yes
CPU reservation of a resource	Yes	No	No	No	No	No
Workload trace-based simulation	Yes	No	No	Yes	No	No
Differentiated network Qos	Yes	No	No	No	No	No
Generate background network traffic	Yes	Yes	No	No	Yes	Yes

Figure 4.1: Comparison of features supported by different large-scale simulators. Taken from [SCV+08].

First, from a functionality point of view, the comparison of features provided in Fig. 4.1 underlines that GridSim provides a full set of features that makes it highly customizable. Moreover, it has a basic QoS support.

From a SoRTGrid perspective, the main advantage is that it is written in Java and its APIs can be inserted in SoRTGrid code in a simple way. The wide variety of possible customizations obtainable by simply extending or configuring the GridSim APIs allow to test any possible application scenario. Regarding SRTS, GridSim provides a management of computational power in MIPS, matching the SRTS job and resource definitions. Similarly, it allows an easy mapping of the Information Component documents in proper internal structures. Finally, it supports Economic Grid and simple cost-models, easily implementable in Owner and User Agents. In the following subsection a deeper analysis of the current porting activity will be presented.

Tools	NS2	GTNetS	OptorSim	SimGrid	GridSim	PlanetSim	PeerSim
Control	Very High	Very High	Very High	Very High	Very High	Very High	Very High
Reproducibility	Perfect	Perfect	Perfect	Perfect	Perfect	Perfect	Perfect
Abstraction	Average	Average	High	High	High	Very High	Very High
Scale	100s	1000s	100s	$10^4 s$	1000s	$10^5 s$	$10^6 s$
Execution time	Equivalent	Equivalent	Faster	Faster	Faster	Faster	Faster
Proc. folding	Often	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
Heterogeneity	Controllable	Controllable	Controllable	Controllable	Controllable	Controllable	Controllable

Figure 4.2: Comparison of experimental characteristics provided by different large-scale simulators. Taken from [GJQ09].

In Fig. 4.3 a comparison on characteristics tied up with experimental variety and customization is provided. Regarding GridSim, it is important to notice that it allows a complete reproducibility of test results and a faster execution time than a real Grid, and provides a high abstraction of the Grid. Moreover, it is written in Java and simulates a Service Oriented Grid, so the porting to GridSim is *simpler* than to the other alternatives. These characteristics make GridSim the best choice for realizing a first logical Grid architecture on which to port the current SoRTGrid prototype implementation.

4.3.1 Mapping SoRTGrid over GridSim for SRTS

This part introduces the guidelines made for porting SoRTGrid implementation over GridSim. Namely, this regards the porting of SoRTGrid architecture on GridSim APIs.

Because in simulating over SoRTGrid there is no deal with an effective middleware, the porting is simpler than those depicted in Fig. 3.11; in fact, it is sufficient to map entities on proper classes of GridSim APIs. In the following paragraph, details on the current porting of SoRTGrid for SRTS are provided.

4.3.1.1 Mapping Resource and Owner side

When dealing with SRTS, the only resource involved is the computational one. GridSim permits to define a list of *Processing Elements* that corresponds to single CPUs offering a given amount of MIPS. A *Grid Resource* is made of a list of processing elements. For my classification [CCM+06], a Single Machine is defined by a resource composed by a single PE, while a homogeneous cluster is defined by a set of identical PEs and an heterogeneous cluster as a set of different PEs.

Resources in GridSim are passive entities. Owner Agents (and User or Facilitator ones) are active. Active entities in GridSim are those involved in simulation. The simulation goes on depending on their interactions. Such entities are implemented as extensions of a *GridSim* class, and so for the Owner Agent to which the previous kind of Grid resources are associated.

4.3.1.2 Mapping SRT jobs and User side

A job in GridSim is called Gridlet and contains the required MIPS for the jobs to complete plus information on file staging, not interesting in our case. A SRT_Job is defined by extending the basic Gridlet with a *time* field for mapping d_n, the amount of time available for completing the job, and an *activation* field. A User Agent is obtained by extending *GridSim* class and adding the list of associated *Gridlet*. A User Agent can add new jobs or remove old ones.

4.3.1.3 SoRTBids and JRMs

SoRTBids and JRMs are implemented as Java classes. Because of the high level of abstraction provided by GridSim there is no need to use, at this step, an XML structure. Such classes have been implemented with field specific for mapping SRTS. In the Owner Agent class, a list of SoRTBids is associated. In the same way, a list of JRMs is contained in the User Agent class. Such lists are initially empty, and during simulation new SoRTBids and JRMs are created as instances of Owner Agents and User Agents. Proper methods for creating automatically such documents, depending on the contingent job requirements (User side) or resource availability (Owner side), have been implemented.

4.3.1.4 Facilitator Agents

Even Facilitator Agents are implemented by extending the GridSim class. Every Facilitator Agent contains a list of associated Owner and User Agents and contains methods for

adding or removing an agent from them, making proper controls (e.g. overbooking). Every Facilitator Agent knows a given subset of neighbors that calls for discovery requests depending on the propagation algorithm used. Moreover, Facilitator Agents contain a structure that acts as a Local Repository where active SoRTBids can be organized through different sorting algorithms. All activities are performed through interaction of Facilitator Agents objects.

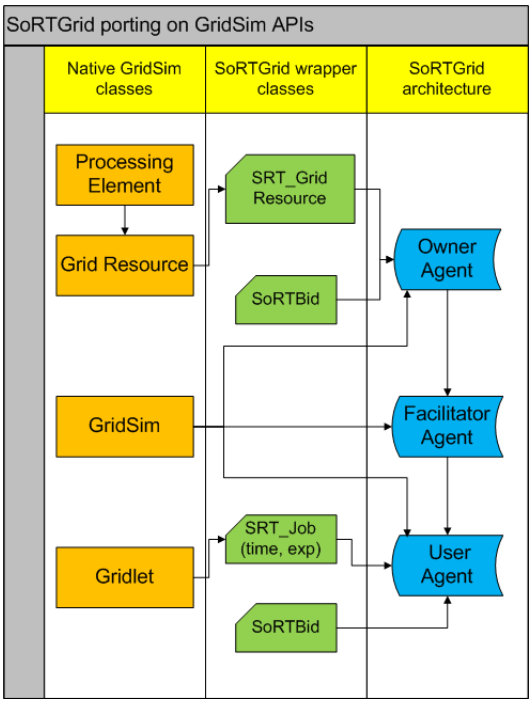

Figure 4.3: Detail on classes for porting SoRTGrid implementation on GridSim.

Currently, the porting activity is quite complete but *configurations* of scenarios and simulations are not yet defined. As a final remark, simulations are supposed to be made on different physical machines through Java RMI technology, in order to manage wide scenarios.

4.3.2 Notes on Grid emulation for SoRTGrid testing

The prototype testing on a Grid simulated by GridSim is sufficient for evaluating the correctness and the global performance of the prototype. However, GridSim constitutes a high abstraction of a Grid (i.e. the Grid is effective but Grid resources are simulated Java structures). In particular, *GridSim* simulates the Grid structures and can be manually configured with latencies but it *does not behave as a Grid middleware* so issues concerning

the interaction with a real middleware can not be investigated. Because SoRTGrid deals with time constraints, it would be an important step to *analyze the impact on performance and degrade caused by latencies proper of the middleware*. More specifically, it is important to evaluate how much the middleware could lower the performance of SoRTGrid prototype in comparison with the simulation over GridSim. For this, as a future work, I prefer to define a further SoRTGrid testing phase by porting the framework to an emulated Grid.

A first idea for emulating a Grid is through Virtual Machines (VM). On every VM, Globus Toolkit 4 can be configured so that a set of VMs can form a Grid. Virtual machines allow to define precisely which percentage of resources of the physical machine to use, so it is possible to control the effective performance of the VM and grant the QoS.

In comparison to the simulation case, the limitation comes from the dimension in emulating a Grid: resources in GridSim are Java objects, in this case are physical resources masked by VM, so testing extremely large Grids requires a distributed approach on many physical machines. As a possible solution, it can be considered the use of **in-situ technologies** like the previous mentioned PlanetLab [pla] for hosting the VMs. However, this work is only in an evaluation phase and depends strictly on the results of the tests over GridSim.

Chapter 5

SoRTSim: a simulator for time constraints analysis on Grid

SoRTSim is a tool used for evaluating SoRTGrid through different complication steps (*configurations* of the scenario variables) that allow to test the behavior of the overlay on different situations regarding time constraints, from simple and regular scenarios to more complex ones approaching a real Grid.

In particular, SoRTSim implements a representation of the SoRTGrid overlay that fulfills the assumptions made in Sec. 4.2.1. Such representation is defined by a set of variables. Trivially, not all possible combinations of such variables values are interesting in order to evaluate the SoRTGrid overlay, so the effective tests made on SoRTSim reduce only to some *configurations* of values that are interesting because respect constraints useful for the evaluation purpose (e.g. the core scenario constraints). Test results are analyzed, in order to evaluate the efficiency of the architectural choices made.

In this chapter, the description of the SoRTSim tool, the set of variables defining the representation of SoRTGrid overlay in SoRTSim, and the metrics used for the test simulations are presented. The definition of the testbed configurations, the executions of tests and the analysis of the results with the metrics defined are postponed to the following chapter.

5.1 SoRTSim model and architecture

SoRTSim aims to simulate locally the behavior of a distributed SoRTGrid architecture. Logical components (User, Owner and Facilitator Agents) are locally mapped into proper data structures and the underlying Grid is not an effective one; conversely, the non deterministic and dynamic behavior of the Grid is simulated by differently configuring such

logical components (i.e. their interaction and behaviors).

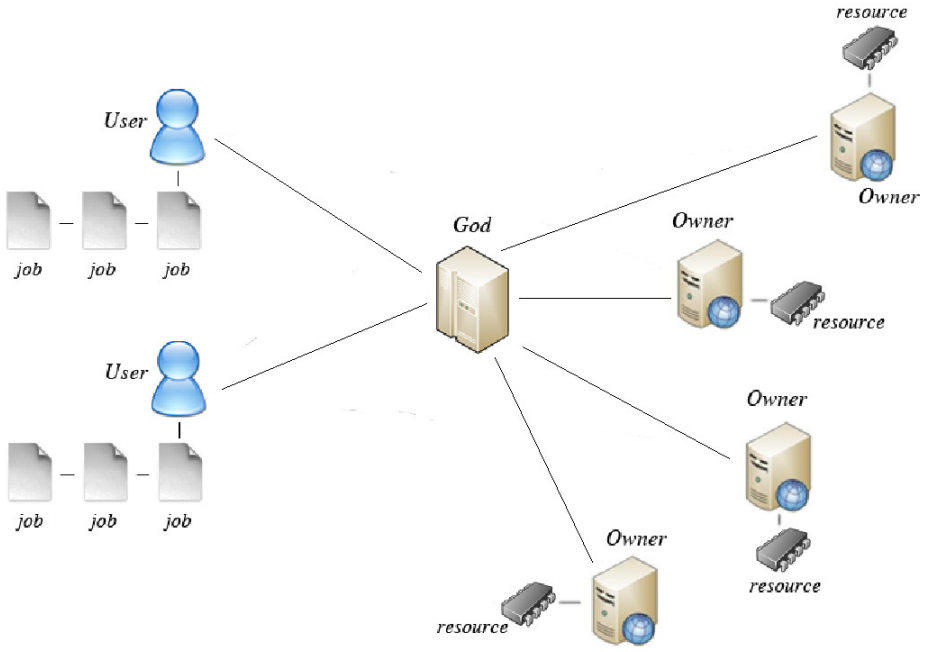

Figure 5.1: *Internal architecture of SoRTSim.*

The architectural model of SoRTSim is represented in Fig. 5.1. User entities are simplified mapping of SoRTGrid User Agents. I do not call them agent because this term indicates entities that are fully autonomous in their behavior and actions. Users in SoRTSim are simpler entities defined by a *behavior*, having a *pool of jobs* to execute. Any User interacts with the middleware (God) in order to check for resource offers suitable for its jobs.

A job is defined by a duration in MI (Millions of Instructions) and a deadline (an absolute instant). At any instant, a job can be inactive or active. Any job in the pool of a User is initially inactive; once activated, it is executed and it can terminate correctly (if completes within the deadline) or in a degraded way (if the deadline is missed). During execution, a job can be considered potentially degraded, but this state depends on the simulation scenario and will be deepened in further sections.

Behavior depends strictly on the kind of jobs belonging to Users. In a real SoRTGrid scenario there are no limits in the definition of behaviors, that can vary during time, being deterministic or non deterministic, and can depend on variables external from the application context (e.g. from policies of the Virtual or Physical Organizations). In SoRTSim, a finite set of deterministic behaviors is defined, in particular for SRTS. Every behavior is defined for testing particular situations and SoRTSim allows to distribute behaviors, randomly or through proper functions, to the set of Users.

The job description, together with the number of desired bids, constitutes the *request* (the equivalent of SoRTGrid JRM) that the User sends to God for requesting a discovery activity among the bids it possesses.

Owners share resources in form of bids and, like Users, they have a defined behavior. In SoRTSim, an Owner manages a single resource and many resources are defined and shared through an equivalent number of Owners. A resource is defined by a fixed amount of *MIPI* (Millions of Instructions Per Instant) and an absolute *expiration time*. A resource is considered active and usable from the instant in which the corresponding bid is uploaded on God repository until the expiration time occurs. While the amount of MIPI remains fixed during any simulation, the definition of the deadline is strategic for the Owner. In fact, its value determines the suitability of the bid for the User side requests. Actually, a User selects (or God selects in its stead) a given resource only if it is sufficient, i.e. if the amount of required MI for the User job is provided by the resource offered within the job deadline. More specifically this means that at the instant k, the resource Res_j is sufficient for the job Job_i if and only if:

$MI_{Job_i} \leq MIPI_{Res_j} * (DL_{Res_j} - k)$

For this, the behavior of the Owner in SoRTSim is oriented to understanding the kind of requests from the User side. In this sense, SoRTSim can support fixed, dynamic and learning-based behaviors.

The central unity is *God* which represents the core process of a SoRTSim simulation. It provides both *simulation* and *synchronization* functionalities.

From the first perspective, it is a projection of a SoRTGrid Facilitator Agent and works as a middleware between User and Owners. It grants the possibility of (1) publishing bids (Owner side); (2) querying for a matching bid (User side). It provides support for selecting bids that match the user request. In this sense, it can work as a proper scheduler or as a partial pre-selector performing the discovery activity as the Facilitator Agents in SoRTGrid. In the first mode, it selects the most suitable bids and can use different policies. In the latter case, it receives the number of desired matching bids from the User and then pre-selects such number of bids (if possible) returning them back to the User for the final selection. A single God can act as a single Facilitator Agent or can even simulate a set of connected ones. This aspect, like the policies used by God, will be investigated successively when dealing with SRTSs.

Besides, God manages the simulation, coordinating and synchronizing all Users and Owners entities and their operations. In practice, it acts as a shared clock between all entities that make operations simultaneously, and governs the passing of instants (e.g. when all activities of the set of entities are terminated). Finally, it constitutes a central point for acquiring and elaborating data collected from entities during simulation. Such functionality is used for calculating different *statistics* on simulation.

As a last remark, SoRTSim supports an Economic mode, featuring a simple cost model (i.e. a budget for Users, a price for the acquired bids and an earn for the Owner who sells successfully the bid). Such mode grants the active state of a User until it has budget to spend. From a simulation perspective, the economy mode in SoRTSim corresponds to a new variable to consider in the policies of entities.

5.1.1 Notes on SoRTSim implementation

The presented architecture and behavior has been mainly implemented with Java Thread technology. Any kind of entity (Owner, User and God) is defined by a proper interface (IOwner, IUser, IGod) containing methods for their supported operations. Entities classes extend the Java Thread class. The choice to define them starting from the Thread class has been necessary in order to obtain an effective concurrent activity of the entities during a single discrete instant. In fact, the only way to allow parallel computation in Java is to use threads explicitly for implementing concurrent objects; since God, Users and Owners require to execute concurrently in order to interact proficiently, the use of Threads is indispensable.

In Java, any extension of the Thread class requires the definition of the method *run()* that is a function containing the concurrent code. Such method contains the behavior and decisional steps that can be made in a single instant by Owner and User entities. In God, the *run()* method is more complex since it manages the whole simulation and the synchronization between User and Owner entities. In particular, it waits for all the entities to finish their *run()* before proceeding on the next instant, and it performs operations concerning the activity of both User and Owner sides. When the simulation ends, it stops all activities and calculates statistics and summarizing graphs. From a practical perspective, User and Owner threads interact with the God one through message exchanging; God directly modifies the status of User and Owner threads for granting the advancement of the simulation.

If active entities have been developed as Threads, passive components like resources and jobs have been defined as stand-alone classes; even in this case, they have been implemented starting from a properly defined interface (IJob and IResource) containing the set of methods characterizing the operation on jobs and resources made by Users and Owners respectively.

The simulation scenario is defined by properly instantiating objects from any class. This operation permits to obtain the definition of the sets of Owner and User entities that compose the simulation. Any entity is built through a proper constructor method invocation. Constructors require suitable parameters for defining the initial state of the entities[1].

[1]The definition of such parameters involves different distributions that will be investigated in 5.2.4

At this point, the simulation starts with the building of the God object that activates all User and Owner ones and coordinates their activity over the instants. More specifically, it manages operations on bids and jobs when an Owner or a User directly invokes a method in the IGod interface and synchronizes the passage of instants. In particular, when a entity completes its activity in the current instant, it suspends its execution as soon as all entities terminate. The *run()* method of any suspended entity is newly activated after the collection of data regarding the instant and the effective passing to the next instant has been completed.

The configuration of scenarios in SoRTSim can be complex, due to the high number of parameters involved. Moreover, for the reproducibility of results and further analysis, it is important to maintain a track of the simulated scenarios. For this, the core simulator of SoRTSim is integrated with a GUI and an XML plug-in.

The first grants an easy way to build-up scenarios, supporting functionalities for automatically building the sets of entities. It maintains all the information concerning the simulation and updates statistic graphs during the simulation execution.

The second extension provides an universal way to save, export and import scenarios and simulation data from and to the SoRTSim simulator.

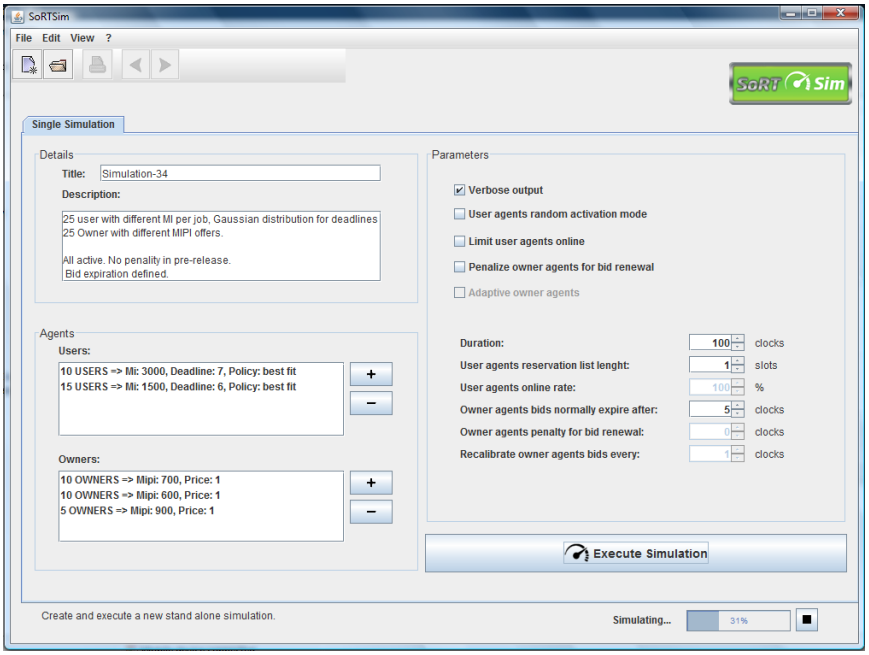

Figure 5.2: *SoRTSim GUI for configuring the simulation.*

Fig. 5.2, 5.3 and 5.4 show examples of the SoRTSim GUI and graphs for evaluating

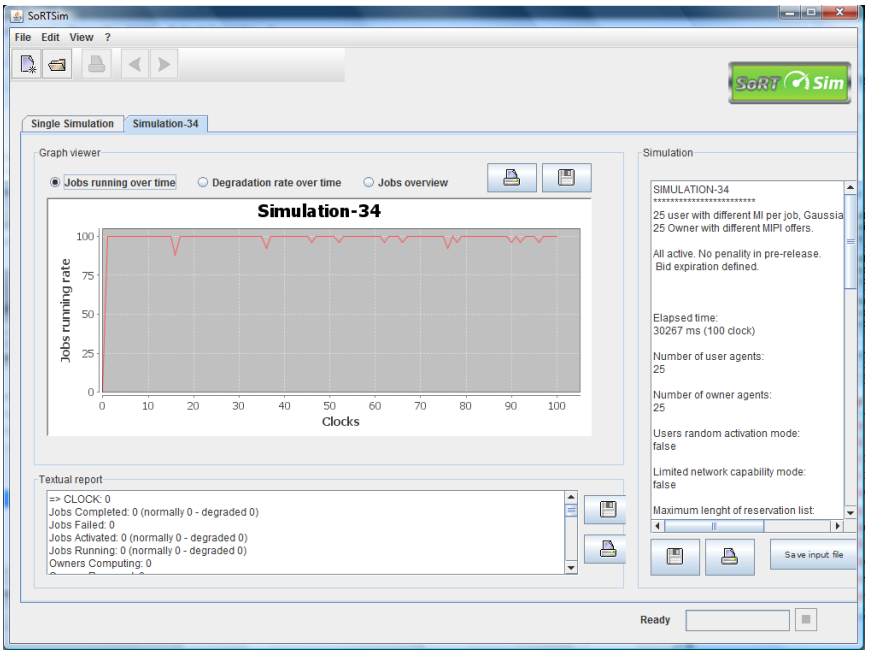

Figure 5.3: *SoRTSim GUI for showing the trend of a statistic over the instants.*

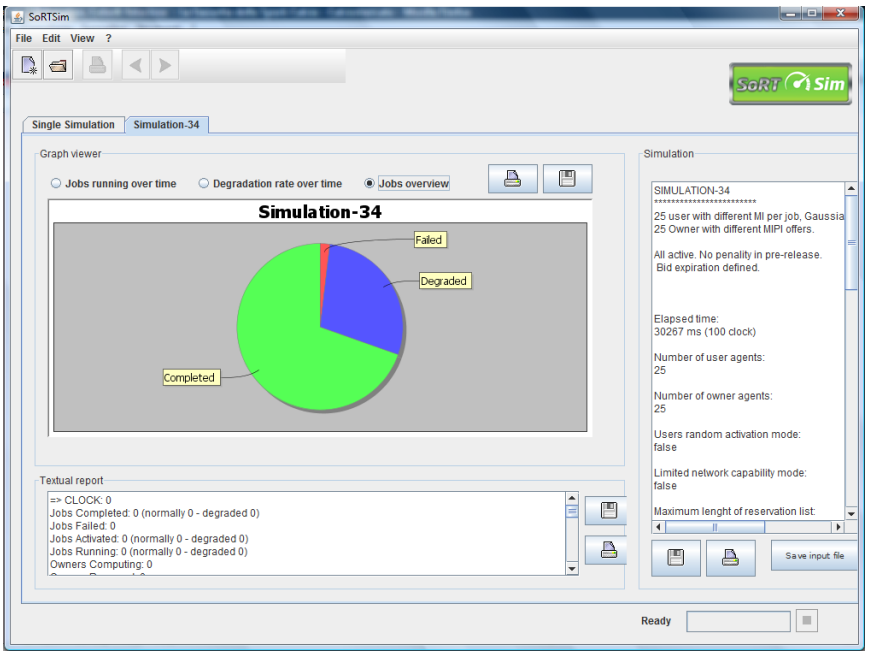

Figure 5.4: *SoRTSim GUI for summarizing the simulation results.*

the trend of the simulation. The considered values depend on the application context. Regarding SRTS, such metrics are explained in the following, after presenting the porting of SRTS on SoRTSim.

5.2 SRTS on SoRTSim

The presented architecture of SoRTSim allows a simple porting of SRTS on it. In this section I analyze how SRTS can be managed on SoRTSim from User, Owner and God sides.

5.2.1 SRTS Jobs and Users

From a simulation perspective, an SRTS job ($J_n =< MI_n, d_n, E_n >$) has some important characteristics. First, the activation instant is unpredictable and the job has to be dispatched on an available resource as soon as it is activated. Second, the duration of the job refers to the execution in the worst case, so it can happen that a job can finish in less instructions. Finally, at every instant there can be only a job instance activated for every User.

These aspects have consequences on the behavior of the User. In any instant, the User has an inactive job instance to manage. If the job activates in that instant, the User is able to correctly execute the job only if it has previously acquired a bid that grants him the immediate access to a sufficient resource. Under these conditions, the User is not able to know a precise deadline until the activation of the job. At the instant t, it can assume that the deadline will be $t + d_n$ if the job activates. If the job does not activate, the deadline changes consequently.

Such situation is manageable by the User only through a ***preventive acquisition*** of resources. This means that the User has to continuously discover and acquire bids in order to dispose always of sufficient resources, in case an instance of a job activates.

Depending on the efficiency of such activity, the User can start in executing the job in different states, once activated. If resources acquired by the User are sufficient for potentially[2] granting a correct execution within the deadline in the worst case, then the job is considered regularly **activated**. Instead, if there are suitable resources but not sufficient for granting the deadline constraint, then the job is dispatched on a resource as **potentially degraded**; in this case, the respect of the deadline depends on the effective duration of

[2]On sufficient resources, the execution is considered to last correctly on fault tolerant resources, if Owners do not violate the contract on the resource.

the job; this is verifiable only by starting an execution and monitoring its evolution during the successive instants. Finally, if there are no resource available, the job is **failed** and dropped without executing.

During the execution of an active job, its status can evolve into two final states: **degraded**, if the execution is beforehand interrupted; this can be caused by the reaching of the deadline or the expiration of the resource reservation before the job completion; **completed**, if the execution terminates correctly and within the deadline on the corresponding resource.

Under this perspective, the activity of a User is reduced to three possible operations:

1. *Resource discovery.* The User checks the state of the acquired resources in order to determine if they are sufficient in case the current inactive job activates. If they are not sufficient, it queries God for new bids. Insufficient resources are released at the end of the instant. The release is notified to God that marks the resource of corresponding Owner as free from the next instant.

2. *Job activation check.* If the current job is inactive, a proper function, based on probability, is called for determining if the job activates or not in the current instant. Different distributions are used during the simulations in order to evaluate different impacts on the final results; however, the User is not aware of the probability value, because SRTS jobs are completely unpredictable in their activation. If the job activates and there are no resource to use for execution, than the job is considered *failed*, and it is dropped from the User queue.

3. *Job termination check.* If the current job is active and executing on a resource, the User checks if the execution completes in the current instant. If so, the current job is considered normally *completed*. Otherwise, if the execution is interrupted (because of deadline reaching or resource expiration) the job is considered *degraded*.

Fig. 5.5 sums up the possible operations and checks that a User can perform within a single instant of the simulation.

Depending on simulation hypotheses, a job can have only some of these states. As instance, suppose to consider jobs with length always equal to the worst case; in this case, the activation of a job when there are no fully sufficient resources would correspond to a failure or a degrade of the job; on the contrary, if there are sufficient resources the job completes correctly. In this case, the potential degradation is not an interesting state.

For the same reason, a User can perform only some of the possible operations in different instants. The choice is guided by the *behavior* of the Users. In SoRTGrid it is impossible to consider any possible specific behavior of the User Agents. Besides, for evaluating the model through SoRTSim it sufficies to consider some meta-behaviors regarding general approaches towards both the request of bids and the selection of resources.

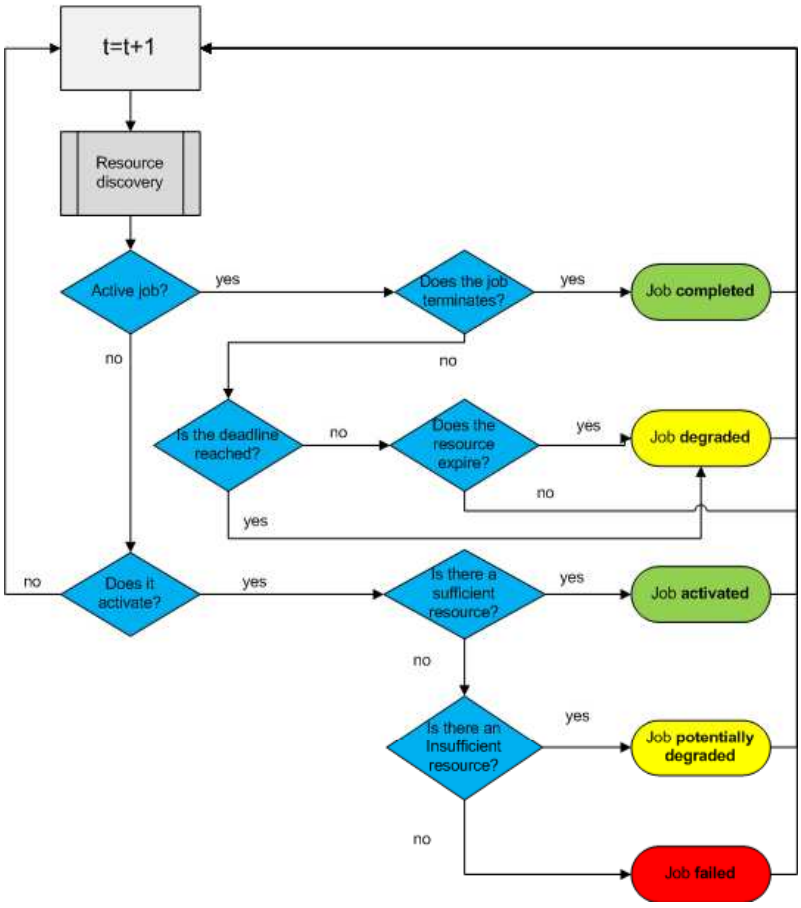

Figure 5.5: *Operations performed by the User for managing SRTS jobs in a single instant of the simulation.*

Currently, there are three kinds of deterministic behaviors for Users that are used for evaluating SoRTGrid on SRTSs:

- *Shrewd*. The User reserves a single resource at most. Until the resource remains sufficient, it grants enough support for the execution of the current inactive job. When the reserved resource becomes insufficient, a new resource is required to God; if a new one is found, then the previous is released. If a new sufficient one is not available the User operates as follows : (1) it releases the previous resource and acquires a new one, in case there is a better resource among the insufficient ones; (2) otherwise, it keeps the current resource. No discovery and selection are performed while the job is active and executing.

- *Speculative*. The User maintains up to two reserved resources at a time. When a resource becomes insufficient it tries to select a new sufficient one. If a new sufficient is not found, it keeps the previous one in any case (e.g. even if there is a better one among the insufficient resources found). Even in this case, no operations related with the acquisition of resources are performed during the execution of an active job.

- *Selfish*. This behavior is used only in Economic mode. The User can select and acquire any amount of resources. Probability is used for defining at every instant if a new resource is to be selected. Distributions used are not random but take into account both the remaining budget and the number of possessed resources. Regarding a resource that becomes insufficient, the choice on starting a new request for bids to God is based on a similar probability distribution, dependent, also in this case, on the number of possessed resources and budget. Even in this behavior, bids are required only while the job is inactive.

There are two assumptions I made on Users and SRTS jobs in SoRTSim: first, any job requires a *single resource only* and the execution is supposed not to be switched on different resources; second, *any User has a single kind of job only* (i.e. the jobs in the pool of a User have the same MI and d_n requirements).

5.2.2 SRTS resources and Owners

Owner side is less variable and complex than the User one. The aim of the Owner is to provide bids that can be suitable for the User side, in order to have its resource acquired as much as possible. This can be guided by different reasons, depending on the context. With economy, more sells mean higher gains for the Owner, while, without economy, the motivation is provided by the higher QoS that has to be granted on resources. In fact, every Owner needs to guarantee total availability of the resource for SRTS purposes;

because this takes to consider the resources fully dedicated to Grid (other uses out of Grid are not allowed, like happens in a typical Grid best effort approach), it is strategic for the single organization that the resource could be acquired as much as possible when offered; otherwise, the effort for assuring a high QoS and availability is useless. In current simulations for evaluating SoRTGrid model, the general approach is without economy; however, economy is considered in few last complication steps.

An Owner with SRTS has to offer bids for a SRTS resource defined by:

$R_m =< MIPS_m, Exp_m >$

In SoRTSim simulations, $MIPS_m$ are considered fixed so the activity of the Owner is reduced to the definition of the expiration of the bid. Owners can upload to God a single active bid at a time[3]. In order to define bid proposals in term of expiration, Owners can consider feedbacks from User choices by evaluating through proper statistical learning algorithms the historical trend of their previous offers; the aim is to find hints for offering more interesting bids than the previous ones.

More specifically, Owners make two kinds of operations:

- *Collect statistical information.* When a bid expires or is released (if acquired), the Owner stores information regarding the bid on a local DB structure; the aim is to build an historical database on the trend of the previous bids, so that it can be used for building better new ones. For SRTS, information regards the proposed expiration, the effective expiration and if it has been acquired or not.

- *New bid creation.* The new bid creation could depend on different variables and can involve statistical algorithms like proper adaptation of K-means and Mean-Shift using the historical DB built during the simulation.

- *New bid upload.* When a bid expires or is released, the Owner works for producing a new bid and upload it to God. The behavior can be different in normal expiration and pre-released case. In the first case, the bid (acquired or not) becomes inactive in the expected moment (Exp_m), so the building of a new bid (both in case of acquisition of the previous or not) has been taken into account; in this case, the new bid is uploaded in the same instant in which the previous expires. If the active bid has been acquired and is pre-released (e.g. because it becomes insufficient for the relative Owner, see 5.2.1), the Owner does not expect to produce and upload a new one in that moment. So in this case, the upload can be shifted to successive instants.

Fig. 5.6 resumes the possible operations of the Owner in an instant.

[3]Because SRTS require 100% availability of the resource, two active bids at the same time is considered overbooking, that is not allowed in SoRTGrid. See 2.5.2.2.

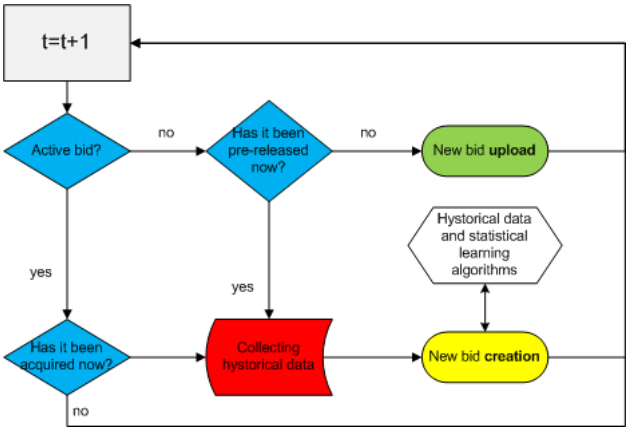

Figure 5.6: Owner's possible operations in an instant.

Like Users, some generic and deterministic behaviors are considered for the Owner:

- *Fixed behavior*. The Owner chooses a fixed period of time for its bids. This means that any bid of the Owner is offered for the same amount of instants for the whole simulation. At the beginning of the simulation, lengths are distributed to Owners through different functions: totally random, ranged random[4], some Gaussian distributions and hash functions. The aim is to evaluate bounds in efficiency on deterministic (and fixed) scenarios during simulation. This behavior is typical of a Physical Organization that has a well defined planning in its participation in SoRTGrid.

- *Variable behavior*. An Owner provides different expirations for different bids. The definition of the new expiration is calculated using functions similar to the previous case, but not related with the User side demand. The aim of this behavior is to find bottlenecks and interesting situations from the Owner side, caused by particular configurations of the resource scenario. This behavior can be typical of a Physical Organization that has not a deep interest in offering high QoS resources and participates in SoRTGrid only when resources are not differently used.

- *Learning-based behavior*. Owners define further bids analyzing the historical trend of the previous ones. This behavior uses statistical learning algorithms (i.e. K-Means and Mean Shift) for analyzing data and defining a publication strategy that takes into account both the duration of the bid and the most suitable instant of publication. Learning-based behaviors can be typical of Physical Organizations that actively participate in SoRTGrid and are interested in proficiently collaborate for some reasons (e.g. gain granted by selling high QoS resources in an Economic Grid).

[4]Random values are extracted in a given interval $[a, b]$.

For now, I do not consider the possibility to remove a published bid before expiration. The aim of this choice is to simplify the Owner side management. In fact, the removal of a previously published bid implies both the use of more complex decisional algorithms and the consideration of new variables (e.g. the time passed after the bid publication, the residual before expiration, ...). Moreover, I do not consider infinite expirations for bids for a simple reason. As remarked, SRTS resources require full availability because of the high QoS constraints that these systems require. For this, an infinite expiration is based on a continuously dedicated resource; this is in contrast with the resource sharing philosophy of Grid where resources are offered (at any QoS level) when not otherwise used by the Virtual Organization.

5.2.3 Middleware management

The middleware is the core component that grants both communication between parts and the control over simulation. The middleware (composed by the God entity) works independently of the kind of application (e.g. SRTS). For this, the functionalities presented here are universal and not typical of SRTSs.

God aims at providing support for the upload and the discovery/selection of bids to User and Owner sides. Because it is universal and not related to the application, its set of operations is fixed and does not depend on different behaviors. At any instant, God receives the bid upload from Owners and a set of requests from Users. For every request, it performs a discovery activity based on the matching degree between the bid and the request.

Fig. 5.7 summarizes the operations carried out by the God entity.

God has a single behavior but the discovery activity can be personalized in different ways.

First, discovery works in two modes: the *selection mode* makes God acting like a metascheduler that chooses the best fitting bid and associates it to the User and the respective Owner as a definitive acquisition. This approach is not typical of a real Grid one and does not support the autonomy of the User side in selecting bids; the *pre-selection mode* makes God work more similarly to the Facilitator Agent of SoRTGrid, selecting the first n bids that match the User request and waiting for a final selection from the User before continuing.

Moreover, by default the discovery is made in a single instant for every request. However, God can be configured so that a selection/preselection activity lasts many instants: such duration can be (1) fixed for the whole simulation or (2) defined randomly or (3) as a function of the current dimension of the Local Repository.

Regarding this last point, God can simulate different God entities connected among them like in an overlay SoRTGrid network. The Local Repository is a single structure but bids

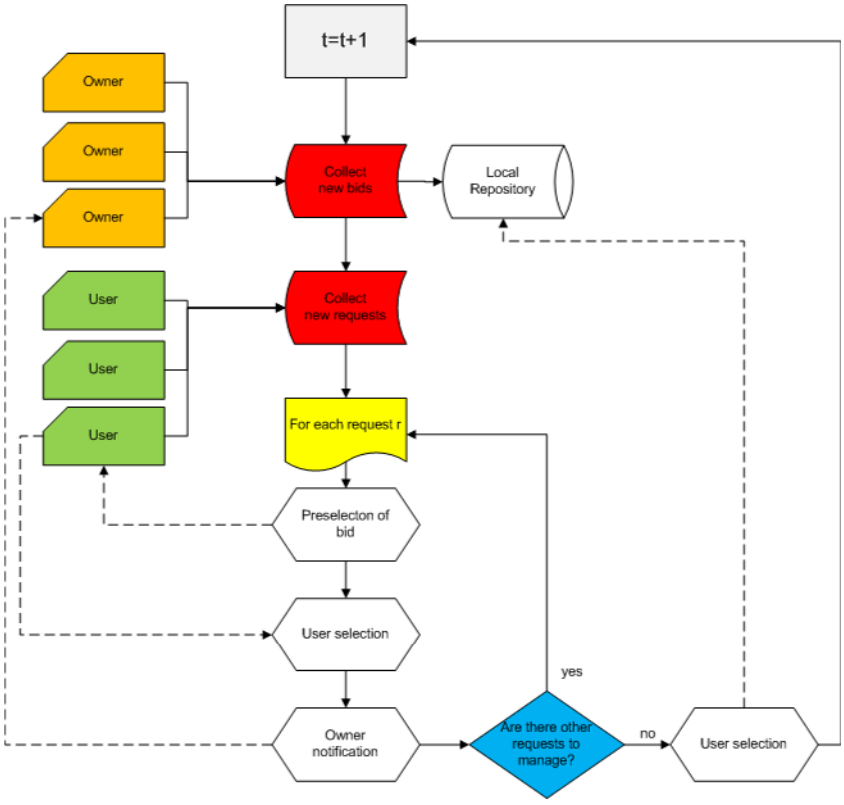

Figure 5.7: *Operations of God entity in a single instant.*

are marked as belonging to a single logical God only. In the same way, Users and Owners are marked. For this, when an Owner upload the new bid, the bid is visible only on the local logical God to which the Owner is registered. In the same way, a User request is managed, at first, as a Local Discovery (considering only bids of its God instance) and it is carried on, in case, as a set of parallel Remote Discoveries with neighbors in successive instants. From the User perspective, this is perceived as a single discovering activity that lasts many instants (compatibly with the User deadline).

Selection, Negotiation and job dispatching operations are atomic and made during a single instant.

5.2.4 Notes on distributions and probability

Some aspects of SoRTSim simulations are based on distributions. In particular, I distinguish between *probability* and *configuration distributions*. Probabilities regard events that can happen during the simulation while configurations regard the definition of the characteristics of the set of entities (Owners and Users).

At every instant, a *probability* value is associated with a single event. Such value is used to determine if the event happens in that instant or not. In the SRTS context, there are three main events that are based on probability:

1. *Entity Activation* - $P(Act)_k$. In scenarios where the set of entities can change, at every instant there is a probability that the entity becomes active (if inactive) or inactive (if active) from the successive instant.
2. *Job activation* - $P(a)_n$. It provides the probability that a pending inactive job activates in the current instant. Because in SRTS there is a single active job at most in any moment, this value is considered 0 during an active job execution.
3. *Job termination* - $P(t)_n$. It provides the probability that the execution of a running job terminates in that instant, lasting less than the maximum duration MI_n. Trivially, when a job reaches the MI_n, it automatically terminates (probability of 100%). In this case, the job termination probability is undefined when the job is inactive.

During a simulation, at every instant and for each event, a random value between 1 and 100 is drawn out, compatibly with the constraints of SRTS (e.g. there is an active job at most at every instant for each User). If such value is lower or equal to the corresponding probability value than the event happens.

For each User, probability values are assigned to the set of the simulation instants through proper distributions. For my aims, I decided to use three kinds of probability:

- *Certainty.* The probability is always the greatest. Concerning the activation of the entities, this means that an entity is always active (it corresponds to scenarios where the set of entities is fixed). Regarding both activation and termination, such distribution allows to consider the worst case in jobs management. In the first case, any job activates as soon as it becomes the current one while, in the second case, any job has always the maximum possible length (MI_n). This distribution helps in evaluating extreme scenarios.

- *Random probability.* The probability is totally random. This is used both for simulating the behavior on very heterogeneous Grid scenarios and for evaluating the influence of casualness in the simulations.

- *Incremental probability.* This distribution increases the probability that the event happens (the job activates, in the activation case, or the job terminates, during the execution) as instants pass. More specifically, if in a given instant k the activation or the termination do not happen, in the successive instant $P(a)_n(k+1) \geq P(a)_n(k)$ or $P(t)_n(k+1) \geq P(a)_t(k)$. When the event happens, the probability comes back to a base value from the successive instant. For the three events based on probability that I consider, it is a good choice. In fact, it is a correct assumption that the probability for a job activation increases during instants in which the job remains inactive. In the same way, it is reasonable to assume that $P(t)_n$ increases with the advancement of the execution. In fact, it is more probable that the job terminates in a number of instructions near MI_n than in a lower quantity.

Probability distributions are involved even in the *Selfish* behavior of the User. In this case, a single incremental probability distribution is used, but it will be presented on the next chapter, during the tests definition.

Configuration distributions are used for configuring the set of Users and Owners. Configuration parameters are four ($MI_n, d_n, MIPI_m, ExpBase_m$[5]) and define the initial state of a single entity. They are distributed on the set of Users and Owners through a pair of *ranged* functions:

1. *Hash function.* This function distributes equally a configuration value in a given range $[a, b]$ on a set of entities. For instance, a hash function, applied to the d_n parameter in a range $[6, 10]$ and on a set of 100 Users, provides 20 random Users with a $d_n = 6$, 20 other Users with a $d_n = 7$ and so on. SoRTSim implements a hash function similar to $(n \ mod \ (b - a)) + a$.

[5]The $ExpBase_m$ parameter is a base value for the duration of the bid and it is used for defining single bid duration, depending on the behavior associated with the Owner.

2. *Discrete Gaussian function.* It distributes the configuration parameter values on different subsets of entities. Given a range $[a, b]$, values near the center of the Gaussian bell (i.e. $a + \frac{b-a}{2}$) are distributed to more entities than those in the sides (i.e. near a or b). SoRTSim implements a Poissonian function as a discrete Gaussian.

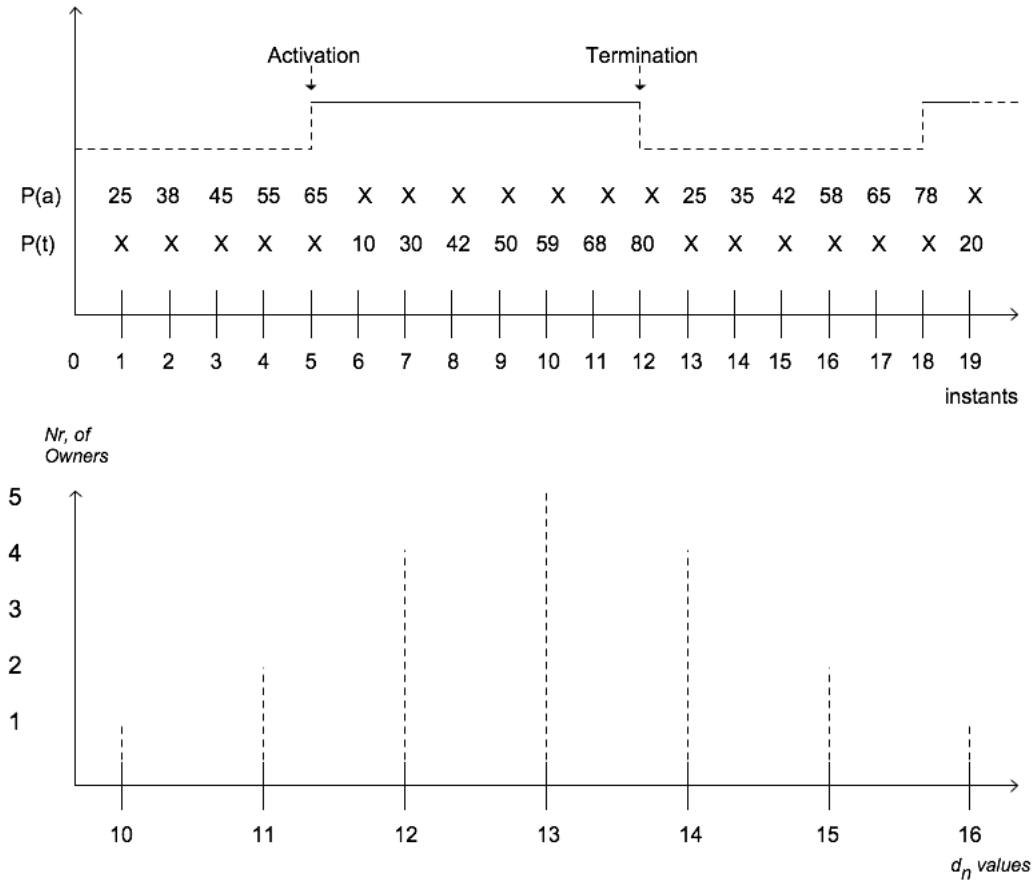

Figure 5.8: *Examples of probability and configuration distributions in a simulation. The first graph shows an incremental probability distribution for the activation and termination of a User job. The second one represents the distribution of a d_n value in a range* $[10, 16]$ *through a Poissonian distribution.*

Fig. 5.8 shows examples of the use of distributions. Depending on the complexity of the scenario analyzed, only some configuration parameters are defined through distributions. As instance, some simple scenarios can have identical d_n for each User or equal $MIPI_m$ for each Owner.

Finally, the hash and Poissonian distributions can be used to distribute Users and Owners

on different logical Gods, but this situation is typical only of the last part of the experiments on SoRTSim.

5.2.5 Definition of simulation metrics

Even if God works independently of the kind of application, the set of statistics it calculates is related to the peculiarities of the jobs. In SRTS, I concentrate on three kinds of metrics. Because I am interested in the behavior of the global system during time, all metrics are global (e.g. on the whole sets of Users and Owners) and dependent on the instant of the simulation.

- *Average degrade - AD.* Average degrade is the main measurement of the efficiency and involves User, Owner and infrastructure sides. At any instant, it provides the percentage of jobs that have been degraded or failed on the total of the activated job until that instant. Such metric depends on both degraded and failed jobs. An higher number of failed jobs means that, in general, there is not a sufficient amount of bids in order to satisfy the number of the User requests; a majority of degraded jobs shows a lack of sufficiency in the proposed bids. The degrade can depend both on suitability of bids and efficiency in selections made by Users and middleware. When the average degree is relevant, it is important to understand which is the main cause. To this purpose, in SoRTSim simulations I consider the following metrics that allow a deeper comprehension.

- *Reservation percentage - RP.* It provides, at any instant, the percentage of published bids that have been selected by the User counterpart (i.e. the acquired resources) on the total of published bids. This quantity measures the efficiency of the choices made at Owner side on the expiration of bids: a higher percentage corresponds to a set of Owners that globally meet the User side requirements; a lower value, together with an average degrade, means that Owner side is not able to meet User requests, because resources remain unselected even if jobs continue to fail or degrade.

- *Real use percentage - RUP.* This metric indicates the effective use (i.e. execution of jobs) of the reserved resources. Such percentage is obtained by the number of instants in which reserved resources execute a job on the total time during which the same resources are reserved. RUP provides a measurement on the efficiency of User selection. A high percentage underlines that Users exploit efficiently the selected resources, wasting few instants of reservation without executing a job; on the contrary, a low percentage highlights a considerable waste of time (and money, in an economic perspective) in comparison with the effective use of the reserved resources.

These metrics are used for evaluating the simulation scenarios. After the definition of the variables that characterize entities and their behavior in SoRTSim simulations, the aim is to define interesting scenarios (i.e. sets of Users and Owners) to evaluate through these three metrics.

5.2.6 Defining incremental experiments

The characterization defined for User, Owner and middleware sides allows to design different scenarios for simulating SoRTGrid overlay. Fig. 5.9 sums up all the variables characterizing the representation of SoRTGrid overlay in SoRTSim and involved in the simulation tests.

Figure 5.9: *Variables involved in SRTS simulation on SoRTSim.*

The definition of the simulations is based on such set of variables and their possible values. More specifically, the complexity of the simulated scenario is related to the number of values that a variable can have and the values distribution among entities that are characterized by that variable. As a practical example, let's consider the set of N Users in a simulation. Every User has infinite instances of a proper job, defined by four variables (MI_n, d_n,

$P(a)_n$, $P(t)_n$); if all Users have different lengths for jobs but identical d_n and *certainty* as probability distributions, the resulting scenario is simpler to evaluate and analyze than in a situation where Users have different values for all the variables. Another example regards the state of entities (active or inactive): if all entities are active during the whole simulation, the scenario is static and proper of a distributed system, while, if the set of active entities varies, the scenario has a dynamic behavior typical of a Grid.

The number of variables allows to build up a high number of possible scenarios. However, as already remarked, it is not aim of the evaluation to test on every possible situation through a quantitative approach; on the contrary, the aim is to concentrate only on *core scenarios*[6] where the total request of MIs from the User side is similar to the full offer of MIs proposed by the Owner side. This takes to avoid simulation scenarios where there is excess of offers in comparison with the Users demand (no degrade) or vice versa (degrade near 100% for trivial reasons).

In the next chapter, both the set of simulations made through incremental complications of the scenario (steps) will be presented (in term of variables domains and values) and results of test made on SoRTSim will be analyzed.

[6]This aspect will be analyzed in the following chapter.

Chapter 6

Evaluating SoRTGrid overlay through SoRTSim

The previous chapter introduced SoRTSim and the way it implements the overlay of SoRTGrid, defining the set of variables involved in the simulation, summarized in Fig. 5.9 and the metrics used to evaluate the simulations results. At this point, it is important to precisely define an *experimental plan*, as a set of *experimental steps*. Any step corresponds to a particular distributed scenario where it is interesting to test the overlay. Any step models a scenario that is more complicated and has less constraints than the previous; as remarked, the idea is to proceed incrementally on scenarios that are always more complex, tending to the complexity of a real Grid environment. The passing from a step to the successive is made after the completion of the experiments and the evaluation of results, even in comparison with results obtained in the previous ones. Following this perspective, I defined *eight* steps in order to evaluate SoRTGrid overlay on SRTSs.

Any step is composed by a set of *configurations*. Given the set of variables, a configuration is composed by a set of values, one for every simulation variable. Configurations in a single step share common values of a subset of variables. Moreover, any configuration corresponds to a single simulation made with SoRTSim (i.e. the configuration is the set of values as input of the simulation).

After defining the experimental plan, tests are made and results analyzed, through the simulation metrics defined, that constitute the output of a SoRTSim simulation.

In this chapter, specifications of experimental steps, configurations, results and analysis on the set of experiments are presented. More specifically, the chapter is composed by three parts and is organized as follows: the first part presents the hypotheses and the planning of the experiments; the second part summarizes the test results, highlighting the most significant ones; moreover an analysis on the experiments is provided. The last part is a

final remark regarding the tests and the differences between the Grid scenario simulated in the last step in SoRTGrid and a real one on which the implementation of SoRTGrid can work on.

6.1 Experiments plan definition

The variables summed up in Fig. 5.9 allow to define an unpredictable number of SRTS scenarios, even if they already constitute a reduction of a general SoRTGrid behavior. Notwithstanding this, the bigger part of the potential scenarios is not interesting for *self-evidence*, *insufficiency* and *feasibility* reasons.

The first case comprehends scenarios that do not provide interesting information on SoRTGrid behavior and have a trivial result. As instance, scenarios with a single User and many Owners are uninteresting because it is trivial to conclude if the User will be satisfied. Insufficient scenarios are made of offers that will never be able to satisfy the Users requests (highest degrade), while the unfeasible scenarios are potentially considerable but the probability that they can correspond to an effective situation is reasonably too low (e.g. infinite offers or too plentiful in comparison with the job requests).

6.1.1 Hypotheses for core scenarios

In order to avoid such uninteresting scenarios, I considered only *core scenarios* that are well-balanced. Such scenarios have to satisfy the following constraints, in order to be considered *core*:

$$M \approx N \tag{6.1}$$

Because every Owner offers a bid per instant at most, and every User submits a single request, a first constraint for balanced scenarios is that the number of active Owners is similar to the number of active Users. This corresponds to have a similar number of offers and requests in the simulation. A consistent difference between N and M takes necessarily to unselected bids ($M >> N$) or unsatisfied requests ($M << N$) for trivial reasons.

For every request, a User needs an average amount of MIPI for completing within the deadline. Such value is provided by dividing the number of total required MI (MI_n) for the amount of time available for completing the job after activation (d_n). In a balanced scenario, it is important that the average of MIPI required by Users corresponds to the average of the $MIPI_m$ offered by the set of Owners:

$$\frac{\sum_{n=1}^{N} \frac{MI_n}{d_n}}{N} \approx \frac{\sum_{m=1}^{M} MIPI_m}{M} \tag{6.2}$$

This hypothesis is sensible in a real context where the autonomous interaction of Users and Owners allows any part to increase the knowledge of the counterpart. This is supposed to take to a convergence of bids toward the requirements of the Users. Such assumption is justified because convergence is the natural evolution of the mutual knowledge of both sides, in order to pursuit the respective aims: bids with too few MIPIs offered require too many instants to complete the job and so the deadline can be missed; on the contrary, bids averagely too high can be wasted or pre-released because they result to be oversized in comparison with the Users requests.

Because SRTS jobs are dependent both on MIPI and deadline, an assumption similar to the previous one has to be done even for time constraints. In fact, the average of the bids duration has to be similar to the average of the deadlines:

$$\frac{\sum_{n=1}^{N} d_n}{N} \approx \frac{\sum_{m=1}^{M} ExpBase_m}{M} \tag{6.3}$$

Such requirement, together with the previous one, grants equilibrium both on the amount of offers and MI requirements and on the duration of bids and deadlines. Even this third assumption is sensible because it is a consequence of the mutual knowledge of the parts.

In general, scenarios that fulfill such constraints are interesting for the evaluation purposes; therefore, the parameters of the steps will be defined so that these three hypotheses are satisfied.

6.1.2 Simulations planning

The core scenario constraints limit the variations of values between the simulation variables. Before defining the experimental steps and referring to Fig. 5.9, I present in details the variable involved in a *configuration* (that corresponds to the characterization of a single simulation). The simulation variables can be divided into three groups, depending on the part of the architecture they are related to. I distinguish the User, the Owner and the God (middleware) variables.

- *User variables.* The User variables are divided into two parts: the part characterizing the global set of Users (two values) and the one defining the single Users (a quintuple). The global set of Users is characterized by $UserSet = (N, ActU\%)$. N is the total number of User entities and $ActU\%$ is the percentage of the total

Users that is active in any instant of the simulation. A single User is defined by $User = (MI, d, P(a), P(t), UBeh)$ defining respectively the amount of MI for any job of the User, the deadline, the kind of probability for activation and termination (Certainty, Random, Incremental) and the User behavior (Shrewd, Speculative, Selfish).

- *Owner variables.* The Owner side is defined by two variables that define the global Owner set and three variables proper of the single Owner entity. The Owner set is defined by $OwnerSet = (M, ActO\%)$, namely defining the number of entities and the percentage of active entities at every instant, similarly to the User case. An Owner is defined by $Owner = (MIPI, ExpBase, OBeh)$, defining the amount of MIPI provided, a base value for the bid expiration that can vary depending on the Owner behavior (defined by $OBeh$). The Owner behavior can be Fixed, Variable or Learning-Based.

- *Middleware variables.* The characteristics of the overlay network are modeled through three variables $GOD = (GMode, GNum, SelPol)$. *Gmode* defines the God selection mode, Metascheduler or Facilitator Agent. In the first case, the selection of the best bid matching the User request is made by the middleware solely. In the second case, the middleware acts as a preselector of N bids, as defined in SoRTGrid, and the best bid selection is made by the single User. $GNum$ defines the number of simulated Facilitator Agents (i.e. the number of God components in the overlay implemented in SoRTSim), while $SelPol$ defines the policy used by the single God to make selections (BestFirst, CheapestFirst); the CheapestFirst policy is used only in an economic mode.

A general *configuration* is then provided by three parts:

User side:

$(N, ActU\%, (MI, d, P(a), P(t), UBeh)^N)$

Owner side:

$(M, ActO\%, (MIPI, ExpBase, OBeh)^M)$

Middleware side:

$(GMode, GNum, SelPol)$.

The meaning of the single variable values can be found in Sect. 5.2. Regarding the set of Users, different values of MI and d can be distributed through the Poissonian or Hash function, in order to define heterogenous User entities in the simulations. Similarly, the same operations can be performed at Owner side, for the $MIPI$ and $ExpBase$ parameters.

On the other hand, all Users and Owners in any set share the same $P(a)$, $P(t)$ and behavior. These three variables peculiarly generate the big amount of non determinism in the simulation; for this, I have chosen to use a common value for all entities in any simulation for the sake of simplicity in the result analysis. In fact, the use of a common behavior or probability distribution helps in determining its impact on the performance metrics in a simpler way than using different behaviors/probabilities. The use of different behaviors or probabilities simultaneously, makes it hard to relate the impact of the single behavior/probabilities to the performance metrics. Scenarios with more diversification are demanded to the prototype testing phase.

Therefore, tests in every step will be centered in checking all possible combinations of single behaviors and probability distributions, in order to evaluate the efficiency of SoRTGrid overlay in term of AD, RP, RUP.

6.1.3 Presenting the experimental steps

An experimental step, as remarked, corresponds to a particular level of complexity in the simulation scenario. Consequently, it puts proper constraints on the variables of the simulation, defining the characteristics of the *configurations* belonging to the step. Before presenting such constraints and motivating any step, I add a last assumption. In any experiment presented in the following, *the number of Users is equal to the number of Owners ($N = M$)*; consequently, for the core scenario hypothesis, *the activation percentage is equal for User and Owner sides* ($ActU\% = ActO\%$). Many simulations have been made with different values for the number of Owner and User, but the most interesting results have been recognized in scenarios with the same number of entities at both parts; for this, I reduce the presentation of the test and the analysis of the results to these cases only.

The aim of this section is to present motivations and details on the different experimental steps. The table in Fig. 6.1 summarizes the general details, defining the characteristics of the configurations of any step.

6.1.3.0.1 1. Dedicated heterogeneous cluster. The simulated scenario is a static heterogeneous cluster, where the set of Owners and Users is defined once and does not change during the whole simulation.

Resources are all identical in their MIPI provisioning but bids can have different durations. The cluster is dedicated to a single kind of application. This assumption takes to consider identical jobs and deadlines for the whole set of Users. The bid selection for a given job is made by God. Preselections of bids by God and consequent autonomous User selections are not considered for this kind of scenario because all Users are identical; for this, a choice made by God on a bid is the same that all Users would make in any instant, having all the

Steps	USER					
	MI	d	P(a)	P(t)	Set of active Users	Behaviors
Dedicated Heterogeneous Cluster	X	X	[C, V, I]	[C, V, I]	Fixed	[Sh., Sp.]
Shared Heterogeneous Cluster	[H, G]	[H, G]	[C, V, I]	[C, V, I]	Fixed	[Sh., Sp.]
Generic Distributed System	[H, G]	[H, G]	[C, V, I]	[C, V, I]	Fixed	[Sh., Sp.]
Dedicated Physical Organization	X	X	[C, V, I]	[C, V, I]	Variable	[Sh., Sp.]
Generic Physical Organization	[H, G]	[H, G]	[C, V, I]	[C, V, I]	Variable	[Sh., Sp.]
Virtual Organization	[H, G]	[H, G]	[C, V, I]	[C, V, I]	Variable	[Sh., Sp.]
Grid	[H, G]	[H, G]	[C, V, I]	[C, V, I]	Variable	[Sh., Sp.]
Economic Grid	[H, G]	[H, G]	[C, V, I]	[C, V, I]	Variable	[Sh., Sp., Se.]

Steps	OWNER				GOD	
	MIPI	ExpBase	Set of active Owners	Behaviors	Mode	N° of logical Gods
Dedicated Heterogeneous Cluster	X	[H, G]	Fixed	[Fi., Va.]	Meta	Single
Shared Heterogeneous Cluster	[H, G]	[H, G]	Fixed	[Fi., Va.]	Meta	Single
Generic Distributed System	[H, G]	[H, G]	Fixed	[Fi., Va.]	FA	Single
Dedicated Physical Organization	[H, G]	[H, G]	Variable	[Fi., Va.]	Meta	Single
Generic Physical Organization	[H, G]	[H, G]	Variable	[Fi., Va.]	Meta	Single
Virtual Organization	[H, G]	[H, G]	Variable	[Fi., Va.]	FA	Single
Grid	[H, G]	[H, G]	Variable	[Fi., Va.]	FA	Many
Economic Grid	[H, G]	[H, G]	Variable	[Fi., Va., L-b]	FA	Many

Figure 6.1: *Experimental steps for evaluating SoRTGrid. Single values for a variable (e.g. Single, Fixed, ...) mean that all the configurations of that step have a single value for the variable. Ranged values for global variables of a set (i.e. $p(a)$, $p(t)$, $UBeh$ and $OBeh$) indicate that the variable can assume different values in the same step. In particular, [C,V,I] indicates that the probability assumes all the three distributions while [Sh, Sp] indicates that in that step only Shrewd and Speculative behavior are considered. The [H, G] range is related to the set of Owners and Users and underlines that both Hash and Gaussian (Poissonian) functions have been used to distribute values for MI, d, $MIPI$ and $ExpBase$ in the set of Users/Owners.*

same requirements that do not change during the simulation.

The aim of this step is to evaluate the management of a single kind of SRTS in a close system, where the global amount of MIPI is sufficient[1] and the efficiency of the system depends on the expiration of the bids. In this step, the competition among Users is strong and converges on a given subset of the available bids (i.e. those who meet their shared requirements); as a consequence of this, bids non belonging to this subset will be always avoided as a first choice because of insufficiency.

6.1.3.0.2 2. Shared Heterogeneous Cluster. This step is an evolution of the previous regarding the User side. The static heterogeneous cluster supports the management of many SRTS. Owners and Users are fixed and continuously active. Users represent different SRTSs (in terms of deadline and MI). The matchmaking between bids and requests is made by God; this situation is typical of a common distributed system with a centralized scheduler (e.g. Condor[MJLM88]). Different parameter values are distributed among Users with the same configuration distributions used for the Owner side.

Differently from the previous step, the variety of User requests is expected to introduce a wider range of selectable bids and less competition (i.e. the group of suitable bids is wider because of User requirements diversification).

6.1.3.0.3 3. Generic Distributed System. This step defines a generic distributed system (i.e. made of a set of clusters) where a decisional point that chooses for the whole set of Users is not considered. On the contrary, selections are made by single Users and the middleware acts as a pre-selector (like the Facilitator Agents in SoRTGrid). Within a single instant, God manages sequentially the Users requests (i.e. it waits for the selection of the User before managing bid belonging to another User), so that the bids preselected by a User can result available for the other Users selections.

The aim of this step is to highlight the differences between the selection made by a single User with a partial knowledge (typical of large distributed systems like Grid) provided by the pre-selection of God and the selections made -at the previous step- by God itself, that has a total knowledge of the set of bids. Another significant difference is that choices made by God are identical for equal User requirements. On the other hand, every User chooses depending on proper characteristics (e.g. the behavior and the characteristics of the acquired bids); from a general perspective over the system, this can take to notice that Users with similar requirements make dissimilar selections on different instants.

[1] For the core scenario constraints.

6.1.3.0.4 4. Dedicated Physical Organization. This step is the first one toward a real Grid scenario. The peculiarity lies in the variation of the sets of active Owners and Users during the simulation. At the beginning of the simulation a set of M Owners, N Users and a percentage value are defined. The percentage value determines the number of entities that are active in each set[2].

At the end of any instant, the status of the entity is checked through the $P(Act)_k$ probability. A value is extracted and, if it is less or equal to the $P(Act)_k$ value, then the entity becomes inactive within the end of the instant. From the successive instant, a random inactive entity becomes active, substituting the inactive one.

From a real context perspective, this represents a typical Physical Organization where the global potential sets of (M) Owners and (N) Users are defined (those belonging to the machines and Users of the PO) but the effective utilization, sharing and demand is dependent on the policies and aims of the single autonomous Owners and Users that freely choose when to become active or inactive. Under this perspective, when a User aims to find resources for its jobs it becomes active and sends requirements; on the other hand, a Owner that has free resources to share becomes active and offers bids until the respective resources have to be dedicated to other users.

In this case, I concentrate on considering a PO specific for a particular kind of SRTS, i.e. a single identical job is managed by the set of Users, for purposes identical to those in the step 1. For similar reasons, I newly consider God as a selection point. Differently from previous cases, since the state of the simulation depends on the variation of the set of Owners and Users, the trend of AD, RP, and RUP could vary significantly during the simulation.

6.1.3.0.5 5. Generic Physical Organization. Like for steps 1 and 2, this step corresponds to the complication of the previous one with the extension to different SRTSs. This means that a set of N Users with different MI and d_n requests is built through Hash and Gaussian functions. In order to consider the impact of many SRTSs on the scenario, I keep on considering God acting as a metascheduler that makes selections; even if this case is various enough to justify a FA agent mode for God[3]. In general, such step represents a general-purpose PO.

In comparison with the previous step, the main difference is expected to be perceived on Owner side because of the variety of the User requests, both for activation/desactivation of Users and for the different kind of requests (i.e. in the previous step, Users were all identical so the difference after a swap between Users is barely perceived). For this, an

[2]Trivially, the percentage is similar for both the sets of Users and Owners because of the constraints in 6.1.

[3]It is postponed to the next step.

Owner can notice a meaningful oscillation on the success of its bid proposals; for instance, the same bid proposed in different instants can take to a selection or a refusal.

6.1.3.0.6 6. Virtual Organization. This step adds the selection of bids to the single User while God acts as a pre-selector of bids.

Even in this case, the differences between the obtained results and the previous ones on identical scenarios permit to evaluate the efficiency of the selections made by single Users, in comparison with choices made by God. This scenario is typical of a Virtual Organization, made of a set of Physical Organizations, where a central metascheduler is not a realistic approach (see Chapt. 2), while a decentralized and autonomous selection is a necessary step.

6.1.3.0.7 7. Grid. This step simulates all the functionalities of SoRTGrid overlay on a generic Grid. In this case, the previous one is complicated by using different instances of Gods that simulate different Virtual Organizations composing a Grid.

The sets of Users and Owners are divided among God instances through the same Hash and Gaussian distributions used before. God instances act as Facilitator Agents (making pre-selections) and are connected through a simulated overlay network that is logically built through the algorithms used in the current SoRTGrid implementation (see Chapt. 3). In this step it is possible to simulate both the local and the remote discovery, by forwarding the request to a (set of) neighbors in the successive instants (compatibly with d_n value).

6.1.3.0.8 8. Economic Grid. The Grid scenario is extended with an economic characterization both on Owner and User side. At Owner side this means that every bid has a price, while at User side any entity has a budget to spend for acquiring bids.

From a simulation perspective, a range for the price of a single MI is defined once and it is the same for all Owners. Any Owner chooses a value in this range. Such value is the price for the single MI for a bid of duration equal to $ExpBase_m$. However, the price of the MI depends on the duration of the bids; this means that the same Owner requires a higher price for a MI in a long-lasting bid than in a shorter one.

A similar approach is followed on User side for defining the initial budget. Such value depends on the MI needed for the execution of a job: Users with longer jobs have a higher budget to spend.

Regarding the activation, Owners continue to depend only on probability while a User becomes definitively inactive when the budget expires.

Behaviors are expanded with the Learning-based (Owners) and the Selfish (Users) behaviors. The economic context is suitable for both these behaviors. In the first case, the use of the behavior is stimulated by the need of competition with other Owners in order to maximize the gain (see Chapt. 2 for further details). In the latter case, the Selfish behavior needs to be limited by a budget cap; on the contrary and out of an economic context, a User has no limitation in its speculation, taking the system to a collapse. Instead, in an economic perspective, the speculation is limited by the risk of becoming inactive because of the depletion of the budget.

Differently from the previous steps, the introduction of Economics as presented before has the consequence that it is possible to analyze a set of *degenerating scenarios* where the set of User entities is reduced during the simulation. In fact, as soon as a User expires the budget, the User is removed from the system; for this, and because any User has a fixed amount of budget, the set of Users is reduced as simulation time passes. On the other hand, the set of Owners never changes.

6.2 Testbed definition and results analysis

In this section, I present the testbeds used for testing the SoRTGrid overlay on different computational scenarios, together with the analysis of the obtained results.

As defined before, any computational scenario corresponds to a proper experimental step and the passing from a step to the follower takes to the building of a new scenario, obtained by complicating the hypotheses of the previous one.

For every step, different testbeds have been defined under the hypothesis of *core scenario* and executed on SoRTSim. Because it is practically impossible to analyze every testbed (and, in some cases, useless), I concentrate in deeply analyzing the most significant testbeds for every step, under a perspective of both simplicity and effectiveness. Moreover, I consider testbeds that are as similar as possible among them, so that a comparison between results obtained in different steps can be consistent and straight.

As remarked, the variables considered in any testbed are the *behaviors* defined for Users (Shrewd, Speculative and Selfish) and Owners (Fixed, Variable and Learning-based) and the *probability distributions* defining the activation and the termination of a job instance (see Fig. 6.1). For every testbed, all possible combinations of the variable values are tested. The evaluation is made on the basis of three metrics (AD, RUP, RP) that in some cases can be extended with others that represent characteristics proper of a certain step.

Anyway, because the choice of probabilities and behaviors determines the trend of the simulation, it is important to point out in which way any variable considered can affect the experimental metrics. To this purpose, before presenting the testbed results of the

eight steps, I analyze the impact of the single variables on metrics, considering a very static scenario that is uninteresting for the SoRTGrid overlay analysis but it is particularly suitable for this purpose.

6.2.1 Impact of probabilities and behaviors on the simulation metrics

The impact of the different variables can be simply evaluated on a fully equilibrated and deterministic scenario. In this case, I consider two sets of Users and Owners with the same cardinality, ($M = N$) where the MIPI offered by any Owner corresponds to the average amount of MIPI required by any User for completing the job exactly within the deadline ($MIPI_m = \frac{MI_n}{DL_n}$) and the User deadline is identical to the duration of a bid ($DL_n = Exp_m$). All User and Owner entities are identical.

The main characteristic of such scenario is that it fulfills constraints that are stricter than the core scenario ones. In fact, all the core scenario formulas loosely bind two quantities through an approximation constraint ($\approx$). It is trivial to verify that in this case the approximation corresponds to a perfect equivalence of the quantities ($=$).

Regarding the simulation variables, the scenario is perfectly balanced with a Shrewd User behavior (any User reserves a single bid at most), a Fixed Owner behavior (bids are all identical and have a duration that coincides with the Users deadline), a Certain (100%) activation probability (i.e. any job activates as soon as it becomes pending) and a Certain termination probability (i.e. the job terminates after executing the maximum number of MI).

My aim is to evaluate the impact of variables values on the simulation by modifying a single variable at time in this balanced scenario and to verify the changes on metrics in comparison with the balanced one.

Before starting the evaluation activity, I define a syntax for naming a simulation so that all information on the variable values used can be quickly understood. To this purpose, any simulation (related to a configuration) is named (both in report and graphs) in this way:

$$< Step > - < U-beh > - < O-beh > - < A-Pb > - < T-Pb > - < Ent-distr > \tag{6.4}$$

where:

- $< Step >$ is the experimental step (from 0 to 8). 0 correspond to this first step for evaluating the impact of the variable values;

- $< U - beh >$ is the User behavior and can assume the values {Sh, Sp, Se} for Shrewd, Speculative and Selfish behavior respectively;
- $< O - beh >$ is the Owner behavior with values {Fx, Va, Lb} for Fixed, Variable and Learning-based behavior respectively;
- $< A - Pb >$ and $< T - Pb >$ indicate the activation and termination probability distributions. Both parameters can assume values in {1, RND, Ixy}. 1 and RND indicate the Certainty and Random probability, while I indicates the incremental probability and xy the base probability value. The increment is fixed and corresponds to 10%;
- $< Ent - distr >$. This is a optional parameter that indicates the distribution used to assign values in a given range to the set of Users (i.e. MI and DL) and Owners (i.e. MIPI and Exp). The values of this parameter are H or G, indicating the Hash and the Poissonian (as discrete Gaussian) distributions. The missing of this value indicates that no distributions have been used in the simulation.

6.2.1.1 Simulation of the balanced scenario

The simulation of the balanced scenario considers 100 Users entities (Shrewd) and 100 Owner entities (Fixed, Exp=10). Any User has 1000 MI to compute, any Owner offers bids that last 10 instants and offers 100 MIPI, perfectly fulfilling any User demand. The Certainty distribution for activation and termination determines that all Users activate a job instance at instant $1 + 10k$ for any k, and in that instant only. In the report of the behavior, it is expected to recognize 100 activations in the $1 + 10k$ instants and no activations in others. For similar reasons, any job completes in the $10k$ instant. I made a 200 instant length simulation; there is no need to exceed in the simulation length because the scenario is fully deterministic. In this simulation and for the consideration made previously, any User correctly terminates 20 job instances.

Fig. 6.2 shows the trend of the metrics during the simulation. It is clear that in this scenario and in every instance of the simulation all metrics reach the best value. The degrade is always 0 while the RP and the RUP is always at 100%. The RP is barely the metric used for measuring the satisfaction degree of the Owner side (which percentage of the total time a bid is effectively reserved). On the other hand, the RUP value represents the User side satisfaction, indicating the percentage of time during which the resource is used on the total reservation time of the corresponding bid.

This scenario represents the ideal status of a SoRTGrid system, where all offering Owners and demanding Users are fully satisfied, without any job instance dropped or degraded.

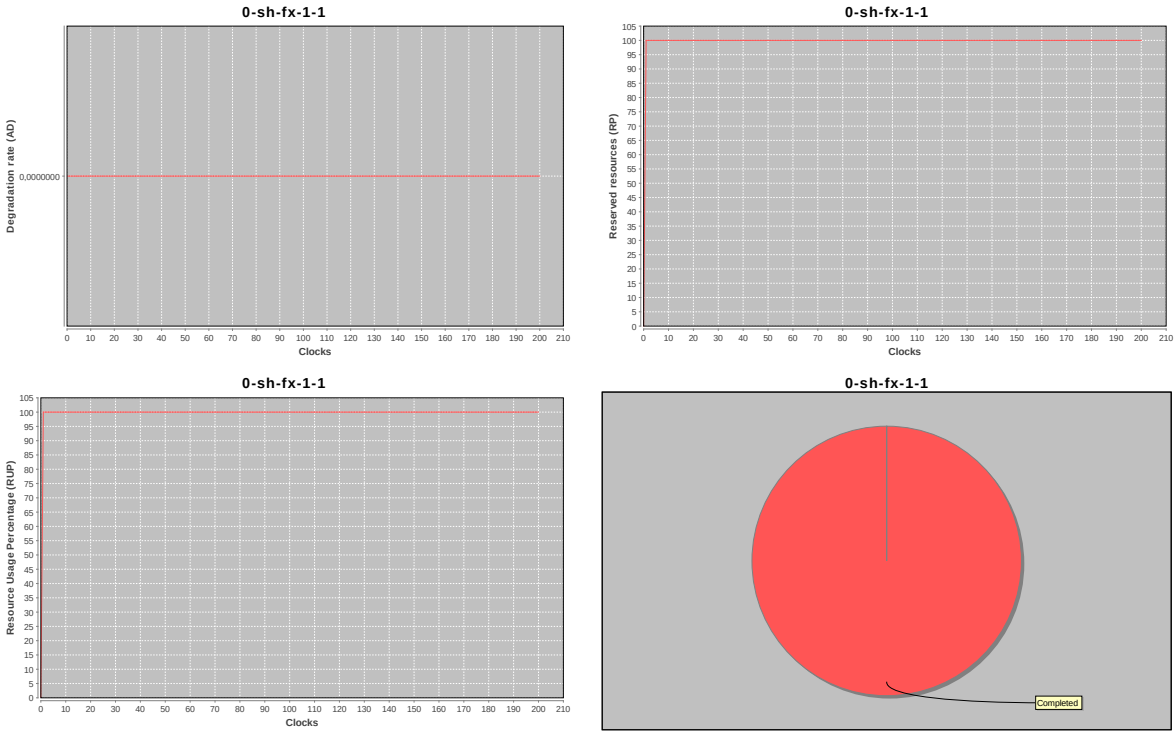

Figure 6.2: Metrics in the balanced scenario.

6.2.1.2 Impact of Activation and Termination probabilities

The activation probability defines if the job activates or not in a given moment. The termination probability determines the effective duration of the job. Both activation and termination probabilities have consequences at User and Owner side.

The activation probability determines when a User effectively starts to use an acquired resource, so the metric that is more dependent on the activation probability is the RUP. At Owner side, the activation probability influences the number of bids that are offered. For instance, if a User acquires a bid that lasts 10 instants at instant 1 and the job (lasting 10 instant) does not activate, than the resource is considered insufficient from the second instant and it is released; this takes the Owner to define a new bid from the instant 2 instead that at instant 11 as expected, forcing to modify its bids publication plan.

The termination probability affects the Owner side in a similar way. A job that lasts less than the expected termination takes to a possible pre-release and new generation of bid. This has consequences in scenarios where there is a latency in producing a new bid after a pre-release (from Step 7). At User side, this creates difficulty in understanding the right amount of MIPI to search for. In an Economic context, this can take to loss of gain or increase of expenses.

The tests made on the balanced scenario using random and incremental probabilities in stead of the certainty one have highlighted a modification only on RUP metrics and in the number of activated jobs. In fact, as all bids suffice the User demand and an Owner renews the bid as soon as the previous is pre-released, the AD is always 0. In the same way, the RP is still 100% because the User, that pre-releases the bid, acquires the new one on the next cycle. Fig. 6.3 summarizes the trend of RUP in different cases. It is important to notice the difference between a random distribution and an incremental one. Although the average RUP at the end of the simulation is similar, the graphs show interesting differences between the two distributions on activation and termination. Both trends are sinusoidal. On activation, both the random and the incremental distributions show the same interval of variation in amplitude (from 45% to 100%) but the incremental one has a regular distribution of peaks. On termination, the peaks distribution is similar in both cases but the incremental one shows a more strict amplitude range and regular trend.

The last two graphs in Fig. 6.3 show the trend of RUP when using the two distributions for both activation and termination. Resulting graphs are quite similar, as the average RUP calculated at the end of the simulation. Because of such results, I choose to analyze only the random probability in the steps from 1 to 7, because there is no adaptation or learning activities on User and Owner side (so the incremental is uninteresting) and no significant differences are recognizable in using one rather than the other. On the contrary, on Step 8, the incremental probability will be used to test the Learning-based behavior of

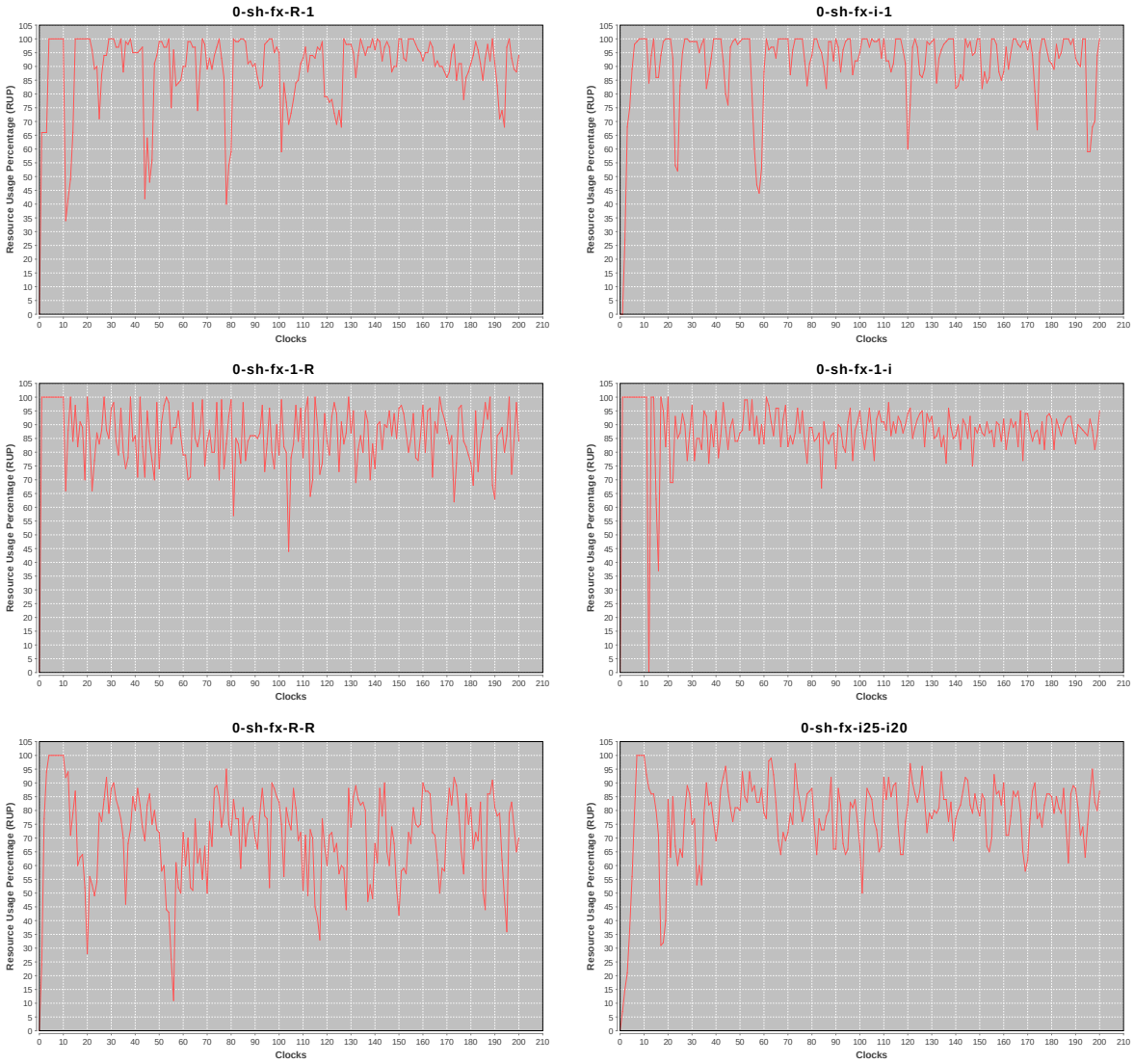

Figure 6.3: RUP trend on random and incremental probabilities for job activation and termination.

the Owners.

Regarding the job activated, the incremental and the random probabilities on activation reduce the number of total jobs because some instants can pass before the job activates. On the contrary, the termination distributions increase such value since the previous termination of a job takes to the pending of a new one. More in detail, the simulation took to an average of 1762 jobs activated with a random probability for activation and to 3845 jobs activated with a random termination probability (the value in the balanced case is, trivially, 2000). When using both distributions, the number of activations is 3004, an average between the two limits.

6.2.1.3 Impact of User behaviors

The User behavior determines in which way the resources of the Grid are acquired. In the balanced scenario, the Shrewd behavior is ideal because a User acquires a single bid per time. This is an advantage from the Grid perspective but can lack in providing sufficient guarantees to the User: in fact, as soon as the bid expires, the User has to start a new discovery and the possibility to execute a job depends strictly on the success of such activity. On the contrary, a Speculative approach takes the User to acquire more resources than those required but allows him to dispose of a sort of reserve in case a resource is not available. In my testbed, this behavior grants the User to acquire two resources at most[4].

To efficiently test the Speculative behavior, it is fundamental to use the random distribution for activation and termination. In fact, in the balanced scenario, if all Users activate and terminate in same instant, any User has a single bid to acquire and no speculation is possible. With speculation, the degrade and failures are expected to raise, because the speculation subtracts bids to other requesting Users.

Results summarized in Fig. 6.4 underline that the test behaves as expected. The most important result is the high number of failures that corresponds to the number of Users that speculate. In fact, for every User that acquires two resources, there is another that is not able to find a suitable bid and the activation of the job fails. The presence of some degraded jobs is still a consequence of the Speculative behavior. The justification is explainable with a simple example. Let suppose that a User reserve two bids in two consecutive instants without activation and then, in the third instant a job activates and terminates; at the end of the execution both resources can be insufficient and the used one is released; the User tries, in the following instant, to select a new bid but it is not able to find one. If the job activates, it is necessarily executed on the insufficient bid and then it degrades.

Considerations made for the Selfish behavior are also valid for the Selfish one. The

[4]As explained, there is the need of a single resource per job for running an execution, in my hypotheses.

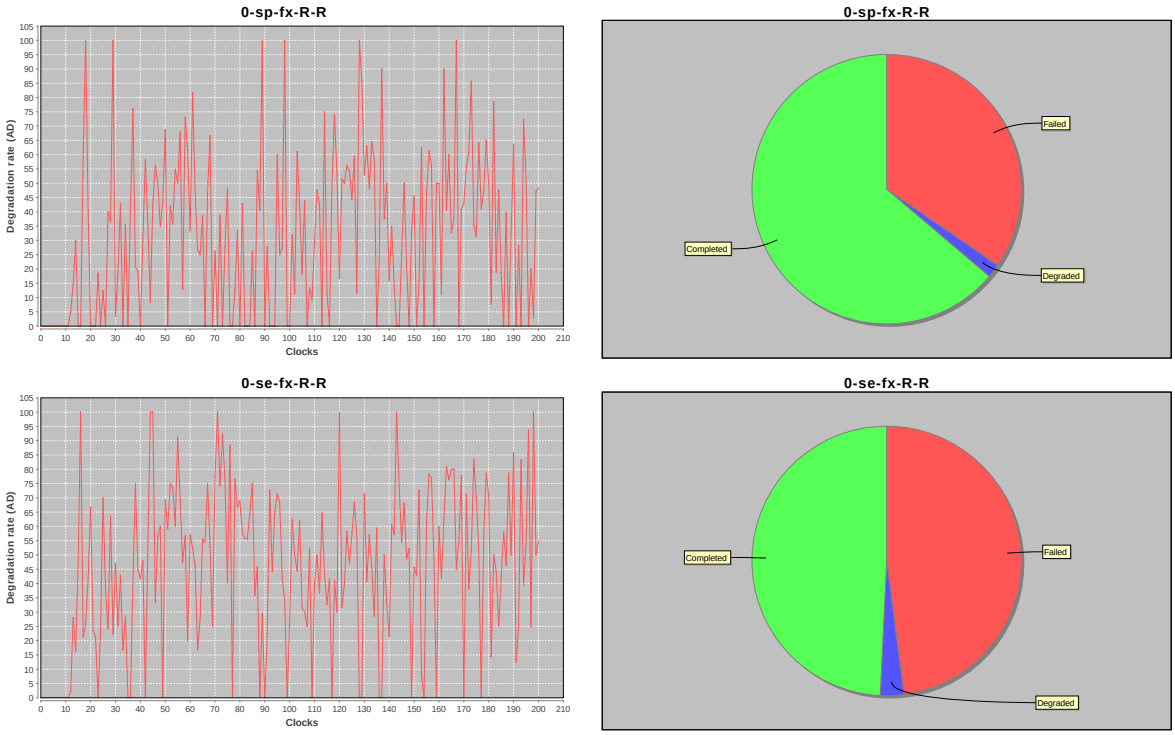

Figure 6.4: Effects of Speculative and Selfish behavior on job degradation and failures.

difference is that some User can acquire more than two bids. This takes to a higher degrade and, in particular, a higher number of failures. The trend of the AD graphs is similar but the Selfish has a higher Average Degree, caused by the big number of failures. As remarked, in case of non economic Grid, the Selfish behavior takes necessarily to degradation and wasting of resources because the User has no limitation in its acquisitions and the system collapses.

6.2.1.4 Impact of Owner behaviors

Owner behaviors define the strategies of the single Owner for defining the bids expiration. The Variable behavior creates bids with a random expiration in the range $[\frac{ExpBase}{2}, 3\frac{ExpBase}{2}]$. The Learning-based behavior is performed keeping track of the trend of the bids offered. Every Owner registers the effective duration of every bid reservation and defines the next bid expiration by evaluating the average of the previous bid durations.

Fig. 6.5 shows the impact of such behaviors on degrade.

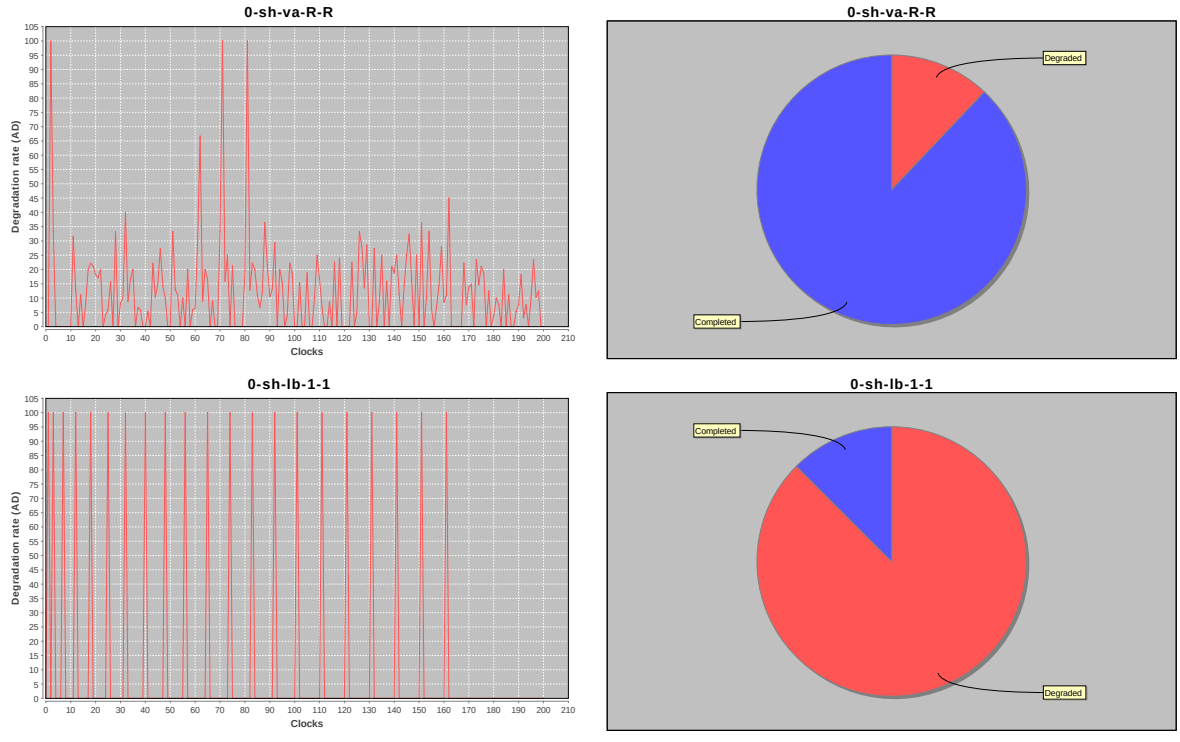

Figure 6.5: Effects of Variable and Learning-based Owner behavior on degradation.

Regarding the Variable behavior, the half of the bids produced are insufficient (i.e. the

duration is less than the deadline requested by Users), for this a 50% degrade is expected. However, the random termination probabilities make some instance to terminate in advance, so the final degrade is lower.

The trend of the Learning-based graph shows how the Owner side adapts to the User side requirements. As shown, the degrade lowers over the instants; in the final part of the simulation the degrade is reduced to zero.

The average degrade at the end of the simulation is considerable higher in the Learning-based case but this is caused by the length of the simulation; a longer simulation will lower the degrade in the Learning-based case (the degree is maintained at zero after the initial learning), while in the Variable approach the average degrade remains globally constant.

6.2.2 Analyzing the experimental results

After presenting the effects of the different behaviors and distributions on the simulation metrics, I analyze which is their impact in scenarios more and more complicated in their characteristics. The main goal, as remarked, is to determine which is the impact of a overlay architecture based on autonomous requests and offers, like the SoRTGrid one, on different scenarios.

The experimental plan has been defined in this way for a twofold reason. First, the evolution of the scenarios by successive and minimal complications is important to understand if the models work well as soon as the scenario tends to the complexity of a Grid. Secondly, the comparison between scenarios that differ by only few characteristics is useful to highlight which specific modifications have the higher impact on the simulation metrics.

Many core scenarios have been tested. However, in the following and for the sake of synthesis, I consider only core scenarios that are as similar as possible to the previous one, because of the simplicity of comprehension they offer.

6.2.2.1 Dedicated Heterogeneous Cluster

In this first scenario, a set of identical Users is tested on an heterogeneous set of Owners with the middleware that makes selection in User's stead, like in a heterogeneous cluster. The difference between the requests and the offers, makes some kind of bids always insufficient for the User set; for this, any kind of activation on such a bid is always degraded. Both sets are identical so no failures are expected without a speculative behavior or if the activation is certain.

The scenario is composed by 300 Users with $MI = 1000$ and $DL = 10$ instants; on the counterpart there are 300 Owners with all $MIPI = 100$ and ExpBase distributed in the

range $[6, 14]$ through hash and poissonian functions. It is trivial to verify that these values respect the core scenario constraints.

Step 1		USER BEHAVIOR							
		SHREWD				SPECULATIVE			
OWNER BEHAVIOR	FIXED			P(t)=100%	P(t)=R/I			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	27,4	89,1	P(a)=100%	Completed %	27,4	90,3
			Degraded %	72,6	10,9		Degraded %	72,6	9,7
			Failed %	0	0		Failed %	0	0
			RUP:	100	85,1		RUP:	100	85,9
			RP:	100	100		RP:	100	100
		P(a)=R/I	Completed %	29,6	89,3	P(a)=R/I	Completed %	25,9	51,2
			Degraded %	70,4	10,7		Degraded %	54,1	5,1
			Failed %	0	0		Failed %	20	43,7
			RUP:	87,2	70,6		RUP:	84,5	58,1
			RP:	100	100		RP:	100	100
	VARIABLE			P(t)=100%	P(t)=RND			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	15,4	84,2	P(a)=100%	Completed %	14,3	12,4
			Degraded %	83,6	15,8		Degraded %	85,7	83,4
			Failed %	0	0		Failed %	0	4,2
			RUP:	100	88		RUP:	99,9	89,6
			RP:	100	100		RP:	100	100
		P(a)=R/I	Completed %	14,8	82,8	P(a)=R/I	Completed %	15,2	52,2
			Degraded %	85,2	17,2		Degraded %	84,8	8,7
			Failed %	0	0		Failed %	0	39,1
			RUP:	88,2	88,2		RUP:	89,8	59,7
			RP:	100	100		RP:	100	100

Figure 6.6: *Test results on a simulated dedicated heterogeneous cluster (Step 1).*

Results in Fig. 6.6 show that **User behaviors have a high impact in static scenarios** with a single kind of User request. In particular, four interesting points regarding the impact of both Variable and Speculative behaviors are detected:

- On a Fixed behavior, the Speculative approach reduces to the acquisition of a single resource at every instant. For this, when a job instance terminates, there is no possibility to make speculation if the job re-activates in the successive instant. In case of random termination some differences on the final metrics result but they are caused by the random probability itself.

- If the activation is random too, the speculation is possible. Any time a User does not activate a job can check and select a bid. Because there is the same number of bids and Users in any moment, the speculation leaves necessarily some Users without a selected bid. In this case, they fail. This trend is evident by the significant number of failures occurring under these conditions.

- Even on a Variable behavior, the speculation reduces significantly the number of completed jobs in comparison to the same cases with a Shrewd User behavior (from 80% to 12%), but with different consequences. If the activation probability is certain,

a high number of degraded is recognized. The motivation is the same as in the previous case; the variability in the Owner behavior takes to bids that can be sufficient or insufficient in different instants of the simulation. Because the activation is certain, the User can acquire a single bid before the new activation and often such bid is insufficient.

- In case of a random activation, the speculation takes to a high number of failures instead of degradation, because many Users speculate on bids while others execute, making those Users unable to find suitable bids when they terminate the execution of the current jobs. The low percentage of degraded jobs is due to the insufficiency of the bids, caused by the Variable behavior.

As a final remark, the percentage of failures in the random speculative cases is very significant (43,7% and 39%). In fact, with a speculative behavior that allows any User to acquire at most two bids, the maximum possible failures percentage is 50%. Both of these values are near this worst case one, so it is clear that in this first scenario the speculative approach has strong negative impact on the performance on a static system with fixed requests. In the same system, the variable approach raises the degrade in comparison with the Fixed one (from 60%-70% to 85%).

6.2.2.2 Generic heterogeneous cluster

This step complicates the previous one with a set of heterogeneous Users, making the cluster general-purpose for different kinds of tasks. The heterogeneity is expected to improve the percentage of jobs completed correctly by lowering the degrade (bids that previously could result always insufficient for the whole set of Users now can be sufficient for some Users and insufficient for others).

In comparison with the previous step, the MI and DL are distributed through hash and poissonian functions in ranges $[900, 1100]$ and $[6, 14]$, respecting the core scenario constraints.

The simulations (Fig. 6.7) shows two noticeable results, in comparison with those obtained at Step 1.

First, **the percentage of completed jobs is higher when dealing with jobs with longer completion** (certainty termination percentage) and a bit lower when the effective duration is random, under a certainty percentage for activation. This is recognizable in every behavior combination. The first increase is significant, while the decrease regards only few percentage points. For this, I concentrate on the first case. The few points difference in the latter case is due to the random generation of the scenario.

Step 2		USER BEHAVIOR							
		SHREWD				SPECULATIVE			
OWNER BEHAVIOR	FIXED			P(t)=100%	P(t)=R/I			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	42,6	83,5	P(a)=100%	Completed %	41,8	84,6
			Degraded %	57,4	16,5		Degraded %	58,2	15,4
			Failed %	0	0		Failed %	0	0
			RUP:	96,9	87,2		RUP:	96,8	86,5
			RP:	100	100		RP:	100	100
		P(a)=R/I	Completed %	43,6	84,3	P(a)=R/I	Completed %	34,2	58,4
			Degraded %	56,4	15,7		Degraded %	45,8	12,6
			Failed %	0	0		Failed %	20	29
			RUP:	85,9	70,2		RUP:	83,3	62,1
			RP:	100	100		RP:	100	100
	VARIABLE			P(t)=100%	P(t)=RND			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	23,7	80,2	P(a)=100%	Completed %	24,2	76,6
			Degraded %	76,3	19,8		Degraded %	75,8	23,4
			Failed %	0	0		Failed %	0	0
			RUP:	98,6	67,5		RUP:	98,7	89,6
			RP:	100	100		RP:	100	100
		P(a)=R/I	Completed %	28,7	82,8	P(a)=R/I	Completed %	24,8	55,3
			Degraded %	71,3	17,2		Degraded %	62	14,5
			Failed %	0	0		Failed %	13,2	30,2
			RUP:	88	88,2		RUP:	83,6	61,1
			RP:	100	100		RP:	100	100

Figure 6.7: *Test results on a simulated generic heterogeneous cluster (Step 2).*

Regarding the first case, the cause is the variation of Users requests. With different requests and bids that are potentially suitable for many Users, it is more probable for a User request to find a resource sufficient for executing in the worst case; for this, the percentage of completed jobs is higher.

The results of the combined Speculative and Variable behavior are particularly significant. In general, the number of completed jobs is considerably increased, in particular with a random termination and certain activation mode. In this latter case, failures are reduced to zero and the completed jobs are increased of 65%. Even in this case, the cause is referred to the amount of possible offers that is raised by both behaviors. The Variable clearly produces heterogeneity in the offers distributing and the Speculative behavior helps in re-introducing available bids. For instance, consider a Speculative User with two reserved resources in an instant. It seeks for sufficient resources as soon as a resource becomes insufficient. If so, it frees the bids even if there are no new available ones, because it has another resource to use. In this case, the bid comes back to the middleware (God) and can be selected (as is or renewed) by another User that has less requirements than the previous one[5].

6.2.2.3 Generic Distributed System

In this step, the complication introduced is the selection at User side with a pre-selection made by God. For every User request God preselects a maximum of 20 suitable bids and the choice is made by the single User, using the same selection policy of God. The scenario is similar to the previous one and a single and big number of bids is returned to the User. For this reason, the aim of this step is not to prove that this way is better than the centralized metascheduler, but that similar results can be obtained through this two-step selection that constitutes the basic architectural choice of the SoRTGrid overlay. The scenario is like a distributed system where every resource is known but different actors select and schedule independently their tasks.

For evaluating the efficiency of this method in comparison with the centralized metascheduler, hypotheses have to be more realistic (e.g. pre-selection with different number of bids requested, at least two ways of selection). This is postponed to the 7th Step (Grid) where the scenario is enough big and heterogeneous for realistically evaluating this feature.

Results in Fig. 6.8 show that the behavior is similar as in Step 2, so that the **Facilitator Agent model acts like the centralized metascheduler under loose hypotheses**. Few percentage variations are registered both in negative and positive and are not related to particular behaviors combinations, but, even in this case, they are due to probability

[5]In the previous step a bid insufficient for a User is surely insufficient for all, because all User are identical.

Step 3		USER BEHAVIOR							
		SHREWD				SPECULATIVE			
OWNER BEHAVIOR	FIXED			P(t)=100%	P(t)=R/I			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	40,1	84,3	P(a)=100%	Completed %	37,1	81,9
			Degraded %	59,9	15,7		Degraded %	62,9	18,1
			Failed %	0	0		Failed %	0	0
			RUP:	96,3	86,4		RUP:	96,4	87
			RP:	100	100		RP:	100	100
		P(a)=R/I	Completed %	40,2	82,3	P(a)=R/I	Completed %	29,9	55,4
			Degraded %	59,8	17,7		Degraded %	48	13,1
			Failed %	0	0		Failed %	22,1	31,5
			RUP:	84,9	69,3		RUP:	78,6	59,9
			RP:	100	100		RP:	100	100
	VARIABLE			P(t)=100%	P(t)=RND			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	21,4	76,7	P(a)=100%	Completed %	20,4	77,4
			Degraded %	78,6	23,3		Degraded %	75,8	22,6
			Failed %	0	0		Failed %	0	0
			RUP:	98,5	87,9		RUP:	98,7	87,9
			RP:	100	100		RP:	100	100
		P(a)=R/I	Completed %	23,4	79,2	P(a)=R/I	Completed %	21,9	54,9
			Degraded %	76,6	20,8		Degraded %	65,4	15,2
			Failed %	0	0		Failed %	12,7	29,9
			RUP:	87,8	67		RUP:	85,3	64,1
			RP:	100	100		RP:	100	100

Figure 6.8: *Test results on a simulated generic distributed system (Step 3).*

and the generation of the scenario.

6.2.2.4 Dedicated Physical Organization

This step aims to evaluate the impact of a heterogeneous scenario that dynamically changes during the simulation on the global performance. To obtain this variability, the complication of this step is to limit the number of active Owners and Users to a maximum percentage (70% of the total entities in equal parts). At any instant, idle Owners and Users are randomly substituted by those inactive.

On the other hand, I removed the complication introduced in the past two steps, reconsidering a set of identical Users and the selection activities performed by God, in order to evaluate the impact of the sole heterogeneity. This scenario is similar to a Physical Organization where entities freely join the Grid (active/inactive entities) but dedicated to a given kind of task only.

The removed complications will be reintroduced in the successive steps. Because the scenario is similar to the first one, I compare these results with those obtained in Step 1.

Step 4		USER BEHAVIOR							
		SHREWD				SPECULATIVE			
OWNER BEHAVIOR	FIXED			P(t)=100%	P(t)=R/I			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	43,9	90,1	P(a)=100%	Completed %	43,6	90,3
			Degraded %	56,1	9,9		Degraded %	56,4	9,7
			Failed %	0	0		Failed %	0	0
			RUP:	96,6	84,5		RUP:	96,6	84,9
			RP:	70	70		RP:	70	70
		P(a)=R/I	Completed %	42,6	91,2	P(a)=R/I	Completed %	39,2	89,2
			Degraded %	57,4	8,8		Degraded %	60,8	9,7
			Failed %	0	0		Failed %	0	1,1
			RUP:	87,3	67,5		RUP:	78,6	52,5
			RP:	70	70		RP:	70	70
	VARIABLE			P(t)=100%	P(t)=RND			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	31,2	78,9	P(a)=100%	Completed %	32,3	76,6
			Degraded %	68,8	21,1		Degraded %	67,7	23,4
			Failed %	0	0		Failed %	0	0
			RUP:	97,7	87,6		RUP:	97,6	84,9
			RP:	70	70		RP:	70	70
		P(a)=R/I	Completed %	35,2	83,5	P(a)=R/I	Completed %	32,9	86,2
			Degraded %	64,8	16,5		Degraded %	67,1	13
			Failed %	0	0		Failed %	0	0,8
			RUP:	84	70,6		RUP:	80,5	53,9
			RP:	70	70		RP:	70	70

Figure 6.9: *Test results on a simulated dedicated Physical Organization (Step 4).*

The experimental results (Fig. 6.9) show that **heterogeneity reduces the impact of**

User behaviors on the degrade. Situations where the impact of a given (or couple of) behavior strongly determines both degrade and failures are no more recognizable. In this case, single behaviors bring advantage to the single entity and do not have strong consequences on the global metrics.

For instance, quite identical metrics have been measured in Shrewd and Speculative behaviors under the same Owner behavior, and no big differences are noticed between Speculative and Shrewd on a Variable Owner behavior (like those measured in Step 1).

The identical values of the metrics are related with both the probability percentage and the Owner behavior. It is recognizable that the obtained values are quite identical for the same Owner behavior, activation probability and termination probability.

6.2.2.5 Generic Physical Organization

This step extends the previous one by reintroducing variability in the set of Users requests. From a practical perspective, I am introducing User requests heterogeneity on the dynamism of the previous scenario. This scenario is similar to a single generic Physical Organization where heterogeneity is high but managed by a centralized metascheduler, because of the reduced number of entities involved.

Results in Fig. 6.10 show that **passing from a specific to a generic PO does not worse the global performance on the overlay**. On the contrary, the completed jobs percentage is slightly raised in comparison with the previous results; this is due to the variability of the Users requests, like it has been already proven at Step 2.

A noticeable increment in the number of completed jobs is obtained for the Fixed behavior for activation and termination with 100% activation and termination probabilities. Even in this case, the increment is due to the differentiated requests of the Users; different requests have different subsets of bids that are sufficient (i.e. all Users do not compete for the same bids), so more Users can complete correctly.

This step is the last one where the centralized metascheduler approach is a good assumption; from the following step both sets of Users and Owners will increase in cardinality, so a centralized approach is not realistic. The aims of the next steps is to evaluate how the centralized selection and discovery process on a big scenario impacts on the global performance.

6.2.2.6 Virtual Organization

From this step, the simulated scenario changes considerably both in dimension and in duration of every single simulation. As remarked, the dimension takes the decisional process

Step 5		USER BEHAVIOR							
		SHREWD				SPECULATIVE			
OWNER BEHAVIOR	FIXED			P(t)=100%	P(t)=R/I			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	60,2	90	P(a)=100%	Completed %	56,9	91,8
			Degraded %	39,8	10		Degraded %	43,1	8,2
			Failed %	0	0		Failed %	0	0
			RUP:	95,3	86,2		RUP:	96	86,2
			RP:	70	70		RP:	70	70
		P(a)=R/I	Completed %	62,4	88,9	P(a)=R/I	Completed %	52,7	89,2
			Degraded %	37,6	11,1		Degraded %	47,3	10,5
			Failed %	0	0		Failed %	0	0,3
			RUP:	85,2	86,6		RUP:	78,6	52,8
			RP:	70	70		RP:	70	70
	VARIABLE			P(t)=100%	P(t)=RND			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	33,3	79,2	P(a)=100%	Completed %	33,2	80,9
			Degraded %	66,7	20,8		Degraded %	66,8	19,1
			Failed %	0	0		Failed %	0	0
			RUP:	98	88,1		RUP:	97,9	87,4
			RP:	70	70		RP:	70	70
		P(a)=R/I	Completed %	35,7	81,4	P(a)=R/I	Completed %	32,7	81,5
			Degraded %	64,3	18,6		Degraded %	67,3	18,4
			Failed %	0	0		Failed %	0	0,1
			RUP:	87,3	70,6		RUP:	83,7	56,6
			RP:	70	70		RP:	70	70

Figure 6.10: *Test results on a simulated generic Physical Organization (Step 5).*

to be necessarily made by the single Users after a pre-selection of the middleware of a given number of resources, like in Step 3.

The middleware has the control over the whole set of bids but is not able to choose for every User because both the number of bids (and Owners) and requests (and Users) is too big. For this, the selection is substituted with pre-selection. The main advantage of the pre-selection in comparison with the selection is that its duration in time can be upper-bounded (as in SoRTGrid specification 2.4). In fact, if a User asks the middleware to make choices in its stead it cannot assume how much time can be necessary to find the best bid. On the contrary, a pre-selection of N bids can be upper-bounded by defining a maximum time for the pre-selection. At the end of the boundary time the retrieved bids are returned, and the User makes a choice between them in an independent way, compatibly with the time constraint of its task.

This step represents a big Virtual Organization with autonomous selections and dynamic scenarios. The pre-selection activity is referable to a first generation metascheduler [FTL+02]. The pre-selection activity is bounded with more realistic and variable values (e.g. various *Nbids* for every User in different instants of time). At any moment and for every User, there is a limit for the bids analyzed by the middleware (corresponding to the time upper-bound on the pre-selection) and the returned results are selected only on such a subset of the total available bids. The expected result is a decrease in the global performance because of the inability to perform a complete discovery by the middleware on a huge amount of bids, so the bids returned are the best retrieved but only on a partial selection of the whole set.

In this step, the Virtual Organization is modeled with 1000 Users and 1000 Owners. This could take the middleware to manage a maximum of 1000 request, 1000 releases of bids and 1000 bids upload at worst in every instant. All parameters (DL, MI, MIPI and ExpBase) are distributed in a wider range; more specifically, DL and ExpBase are distributed in $[11, 29]$, MIPI in $[80, 120]$ and MI in $[800, 1200]$. Because of such dimension for a single centralized metascheduler, I suppose that for every request the middleware could analyze only 200 bids for any pre-selection request and returns back to the User a maximum of 10 alternatives.

Results in Fig. 6.11 show the **negative impact of a partial knowledge on the performance** of the whole system. In comparison with figures of Step 5, there is a significant reduction of the completed jobs percentage. This loss is counterbalanced by a high increase of degraded job and a little growth of the failed ones. This consequence is explainable with the partial knowledge that raises the possibility that the obtained bids are not the best ones available, since only a part of the total number of bid is analyzed. Even failures are related to the same cause (i.e. the discovery returns to the User only bids that last the next instant and are not selected, so the activation takes to a failure).

Step 6		USER BEHAVIOR							
		SHREWD				SPECULATIVE			
OWNER BEHAVIOR	FIXED			P(t)=100%	P(t)=R/I			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	19,2	70,2	P(a)=100%	Completed %	17,4	71
			Degraded %	70,3	23,6		Degraded %	72,4	22,6
			Failed %	10,5	6,2		Failed %	10,2	6,4
			RUP:	98	86,2		RUP:	98	87,9
			RP:	70	70		RP:	70	70
		P(a)=R/I	Completed %	27,7	75,6	P(a)=R/I	Completed %	27	80,6
			Degraded %	64,5	20,6		Degraded %	68	17
			Failed %	7,8	3,8		Failed %	5	2,4
			RUP:	85,7	69,6		RUP:	82,8	53,9
			RP:	70	70		RP:	70	70
	VARIABLE			P(t)=100%	P(t)=RND			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	11	65,3	P(a)=100%	Completed %	9,6	63,9
			Degraded %	78,9	27,6		Degraded %	79,9	28,8
			Failed %	10,1	7,1		Failed %	10,5	7,3
			RUP:	99	89		RUP:	99,1	89,4
			RP:	70	70		RP:	70	70
		P(a)=R/I	Completed %	15,5	71,4	P(a)=R/I	Completed %	15	71,2
			Degraded %	76	23,4		Degraded %	79	23
			Failed %	8,5	5,2		Failed %	6	5,8
			RUP:	87,8	69,7		RUP:	85,7	57,8
			RP:	70	70		RP:	70	70

Figure 6.11: *Test results on a simulated Virtual Organization (Step 6).*

Comparing the results with those of Step 5, it is noticeable that the negative impact of the partial knowledge is bigger with the certainty probability distribution. Scenarios with 100% probability for both activation and termination register the worst impact (i.e. till 40% of reduction of the completed job percentage); certainty on activation affects metrics more negatively than certainty on termination. The negative impact is reduced when both activation and termination are random, because they raise the probability that a select bid could be sufficient (i.e. if the duration is less than the worst case) and provide more discovery activities for bid selection before the job activates (i.e. bids can be selected as soon as the job activates).

The experiments made at this step underline that the knowledge level of both parts has to be potentially increased on big scenarios, in order to raise the performance of the system. This takes to scale the same number of entities on different interconnected Gods (Facilitator Agents), so that any God could pre-select in parallel a number of suitable bids through a potentially complete analysis of the active bids.

6.2.2.7 Grid

In this step I simulated a Grid composed by a number of Virtual Organizations. Differently from the previous step, now each VO has a reduced number of Owners and Users (i.e. the set of 1000 Users and Owners is distributed among the Virtual Organizations), so that each VO is able to analyze the whole set of bids for performing a pre-selection. I apply the SoRTGrid architecture (i.e. the overlay network of Facilitator Agents) in this way: in each Virtual Organization there is a middleware that acts as a Facilitator Agent. A given number of Users and Owners are registered to each Facilitator Agent that makes pre-selections for its Users requests by analyzing the bids offered by its Owners and sending the request to other two randomly chosen Facilitator Agents (neighbors) so that the discovery is performed in a parallel way on three subsets of different bids (i.e. remote discovery of SoRTGrid[6]). I used six Facilitator Agents in this experiments, randomly connected to some of the others. The requests from a neighbor are managed as they come from inside the Virtual Organization. I can assume that a Facilitator Agent can manage even requests coming from outside the Virtual Organization because it deals with only a reduced part of the total entities (a sixth in this case, an average of 167 entities per part, lower than the 300 I used to test the centralized discovery in Step 1 to 6).

Low latencies for performing the request to neighbors and for retrieving the results have to be considered. As the implementation of such latencies explicitly on all entities would have been too tricky and require hard modification of the code (SoRTSim is a multithread simulator) without any guarantees on the correctness of the implementation after such modifications, I preferred to simulate these latencies by introducing a delay in the renewal

[6]The local discovery have been extensively tested from Step 1 to Step 6

of a bid (e.g. 1 instant). Such delay can correspond to the time penalty for a remote discovery.

The result expected is *a significant increase of the completed jobs because of an increase of the knowledge within the same time period, through parallel requests to different Facilitator Agents*, even with the bids renewal penalty.

Step 7		USER BEHAVIOR							
		SHREWD				SPECULATIVE			
OWNER BEHAVIOR	FIXED			P(t)=100%	P(t)=R/I			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	81,2	93,8	P(a)=100%	Completed %	80,5	93,3
			Degraded %	17,7	5,6		Degraded %	14,7	4,9
			Failed %	1,1	0,6		Failed %	4,8	1,8
			RUP:	93,3	84,8		RUP:	93	84,7
			RP:	35	35		RP:	70	70
		P(a)=R/I	Completed %	87,8	94,2	P(a)=R/I	Completed %	88,1	91,9
			Degraded %	11,4	5,3		Degraded %	8,2	1,9
			Failed %	0,8	0,5		Failed %	3,7	6,2
			RUP:	62,3	47,5		RUP:	64,3	51,34
			RP:	47,1	53,6		RP:	89,7	93,5
	VARIABLE			P(t)=100%	P(t)=RND			P(t)=100%	P(t)=R/I
		P(a)=100%	Completed %	76,9	91,3	P(a)=100%	Completed %	62,3	85,9
			Degraded %	20,5	7,2		Degraded %	31,1	10,7
			Failed %	2,6	1,5		Failed %	6,6	3,4
			RUP:	98	87,9		RUP:	94,6	85,6
			RP:	70	70		RP:	70	70
		P(a)=R/I	Completed %	84,2	65,5	P(a)=R/I	Completed %	72,1	86,3
			Degraded %	14,7	28,4		Degraded %	22,6	8,7
			Failed %	1,1	6,1		Failed %	5,3	5
			RUP:	63,1	94,4		RUP:	70,5	54,3
			RP:	46,4	69,7		RP:	83,3	90,4

Figure 6.12: *Test results on a simulated Grid (Step 7).*

The experiments highlight a very interesting result. *The average degrade (AD) metric shows the better values registered in the whole set of steps.* The percentage of completed jobs is higher than before in every case and with every combination of behaviors.

The RUP percentage, that provides a measure on how much the User effectively uses the selected resources, varies from case to case but it is always over the 50% (averagely 65%). Because the main issue of a User in a SRTS is to maximize the number of time-constrained jobs correctly completed within the deadline, a loss of reservation time has been expected, even because the resource discovery and acquisition have to be done before the job activates, for granting the maximum coverage in case of the happening of the unpredictable activation (SRTS). In general, the effective usage of more than 50% of the resource is a good result, under this perspective.

The RP percentage, that provides a measure of the Owner satisfaction, is higher in a

Speculative User approach and lower in the Shrewd one, showing a trend similar to the RUP one, granting an average of 60% of satisfaction to the Owner counterpart (and a minimum around 50%).

Because the main metric that measures the effectiveness of a time-constrained system (AD) is particularly low, this last experiment (in comparison with previous ones) determines that **the SoRTGrid overlay improves the management of time-constrained jobs (like the SRTSs ones) on big, distributed and heterogeneous scenarios referable to Grids**.

6.2.2.8 Economic Grid

Because in the motivations of the SoRTGrid model in the previous chapters I argued that the SoRTGrid model is more realistic if applied to an Economic Grid, for the sake of completeness I tested the model on a simulated scenario that has the required economic characterization.

Beyond the model evaluation, this step is important for testing some corollary aspects that are related to economy. First, I argued that the collaboration between the two sides, naturally stimulated by an economic market based on bids (User buys, Owner sells), could take to equilibrate the satisfaction of both parts. To this regard, the Step 7 showed that the SoRTGrid model, out of an economic context, increases the satisfaction of User but provides a lower satisfaction level for the Owner in term of acquired (reserved) resources. For this, at this step my aim is to test if the introduction of the market and the selection on bids based also on cost policies could take to a more equilibrium in the satisfaction of both parts (i.e. lowering a bit the number of jobs completed but raising the RP percentage).

Moreover, it is interesting to evaluate some degrading scenarios (i.e. scenarios in which some entities leave the Grid, without introducing new ones) in comparison to the never-changing scenarios considered till now. The economy helps in this sense: every User is provided by a fixed initial budget; once the budget expires the corresponding User entity leaves the simulation.

Finally, the economic context is good for testing the Learning-based Owner behavior (justified by the need to sell) and the Selfish (limited by the budget) behavior. In the first case, the Owner keeps a historical track of the effective utilization of its bids and offers the successive one with a duration that is the average of the registered values; in this case the learning regard the duration of the bid, while the MIPI is fixed. In the second case, the User acquires as many resources as possible (one per each instant in which it does not execute) until the budget expires and then leaves the simulation.

In this step, any User is provided with an initial budget that allows to acquire averagely 40 sufficient resources for executing job instances. In this way, the Users are expected to

Step 8		USER BEHAVIOR										
		SHREWD				SPECULATIVE				SELFISH		
OWNER BEHAVIOR (Fixed)	p(a)	p(t)	100%	RND	p(a)	p(t)	100%	RND	p(a)	p(t)	100%	RND
	100%	C %	74,6	92,3	100%	C %	74,9	91,9	100%	C %	74,1	92,1
		D %	19,5	5,2		D %	19,2	5		D %	20,1	5
		F %	5,9	2,5		F %	5,9	3,1		F %	5,8	2,9
		RUP	93,5	84,4		RUP	93,5	84,2		RUP	93,6	84,1
		RP	61	60,7		RP	61,3	60,3		RP	61,6	60,3
	RND	C %	84,9	94,2	RND	C %	86,5	92,6	RND	C %	81,5	86,5
		D %	11,8	5,3		D %	10,1	2		D %	8,9	1,8
		F %	3,3	0,5		F %	3,4	5,4		F %	9,6	11,7
		RUP	83,1	47,5		RUP	68,5	58,8		RUP	66,3	39,7
		RP	60,7	53,6		RP	62,3	61,2		RP	61,9	62
OWNER BEHAVIOR (Variable)	p(a)	p(t)	100%	RND	p(a)	p(t)	100%	RND	p(a)	p(t)	100%	RND
	100%	C %	61,2	85,5	100%	C %	59,9	84,6	100%	C %	60,3	83,9
		D %	32,1	10,7		D %	32,3	10,9		D %	32	11,5
		F %	6,7	3,8		F %	7,8	4,5		F %	7,7	4,6
		RUP	94,3	85,1		RUP	94,5	85,4		RUP	94,5	85,5
		RP	60,7	60,3		RP	61,4	61		RP	61,4	61,3
	RND	C %	71,1	90,2	RND	C %	74,2	86,2	RND	C %	68,2	82,1
		D %	24,1	7,2		D %	24	6,2		D %	24,3	8,7
		F %	4,8	2,6		F %	1,8	7,6		F %	7,5	9,2
		RUP	86,4	69,6		RUP	69,4	60,6		RUP	70,2	54,3
		RP	60,3	60,2		RP	61	61,7		RP	61,9	61,4

Figure 6.13: *Test results on a simulated Economic Grid with Selfish Users (Step 8).*

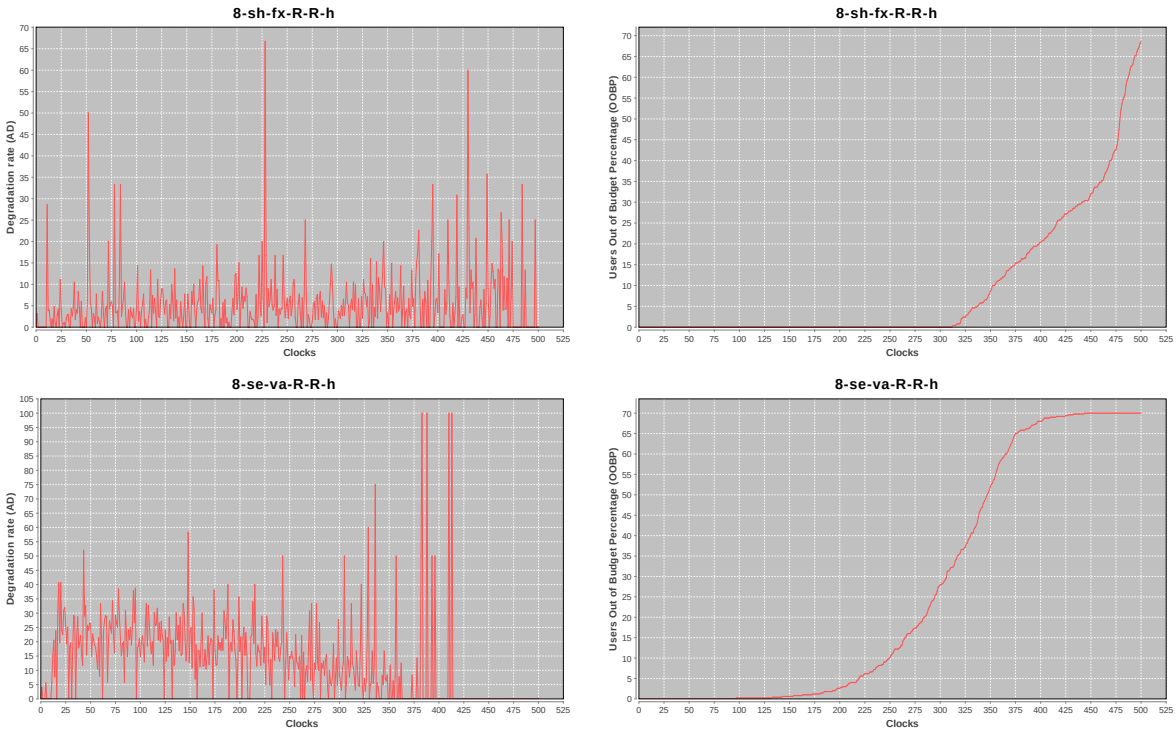

Figure 6.14: Impact of Selfish behavior on the degradation of the scenario and average degrade, in comparison with the Shrewd one.

leave the simulation from the instant number 300.

Results in Fig. 6.13 show that the percentage of completed jobs is slightly lowered in comparison to the previous step. This is due to the cost policy used by the User that takes it to select the cheapest sufficient bid obtained by a pre-selection, in order to extend its permanence on the Grid. This takes to select bids that can become insufficient in few instants and the successive discovery does not grant a satisfactory result. In general, the number of discovery requests is raised owing this policy.

Moreover, the economic approach does not rises significantly the owner satisfaction (RP value) but it levels such value among the different behaviors (it is always around 60%). The cause of the reduced value is that the pre-released activity of the User has impact only on the Owner (that pays the delay for renewing a bid). In a balanced market model the pre-release should impact even on the User (e.g. on budget or on delays in further discoveries), but this aspects have to be investigated in further studies. For similar reasons, the sole consequence that the Selfish behavior has on Users is that they at some time leave the system, but until that instant, the satisfaction of the User is equivalent to the other behaviors. Also on this case, further studies are required.

Fig. 6.14 shows the impact of the Selfish behavior on the degradation of the system and average degrade. In the first case, the comparison of the graphs on the right column demonstrates that a Selfish behavior takes to a rapid collapse of the system; this result, compared with the figures in Fig. 6.13 that do not show a consistent increase in the User satisfaction with respect to Shrewd or Speculative one, underlines the **uselessness of a Selfish or greed approach on economic Grid for managing time-constraints**.

Lastly, the evaluation of the Learning-based behavior reported in Fig. 6.15 takes to two considerations. First, the percentage of completed jobs is averagely raised; this means that the learning algorithm works efficiently (even if it is simple), producing a slight increase in the User satisfaction. In the same way, the RP percentage is slightly increased too, showing that the Learning-based behavior increases even the Owner satisfaction.

However, this result can be interlocutory because the learning algorithm is too simple. A open issue is to repeat the full suite of experiments made at Step 8 by using a proper adaptation of more sophisticated and well-known learning algorithms like K-Means or Mean-Shift.

Step 8	USER BEHAVIOR											
	SHREWD				SPECULATIVE				SELFISH			
OWNER BEHAVIOR (Learning-based)	p(a)	p(t)	100%	INC	p(a)	p(t)	100%	INC	p(a)	p(t)	100%	INC
	100%	C %	76,3	93,3	100%	C %	75,3	92,5	100%	C %	77,2	92,8
		D %	21,3	4,1		D %	17	6,3		D %	20,1	5,6
		F %	2,4	2,6		F %	7,7	1,2		F %	2,7	1,6
		RUP	85,2	84,8		RUP	89,2	54		RUP	62,9	63,5
		RP	61,3	61,7		RP	63,4	62,8		RP	62	62,8
	INC	C %	87,2	92,3	INC	C %	85,2	92,4	INC	C %	82,3	84,2
		D %	10,2	7,1		D %	11	7,2		D %	17,3	5,3
		F %	2,6	0,6		F %	3,8	0,4		F %	0,4	10,5
		RUP	84,6	50,2		RUP	42,3	52,5		RUP	74,2	50,2
		RP	64,3	70,1		RP	62,3	62,1		RP	61,5	63,6

Figure 6.15: *Test results on Learning-based Owner behavior in an simulated Economic Grid (Step 8).*

Finally and from a global perspective, the experiments of this step show that *the SoRTGrid model works efficiently even on an Economic Grid context.*

In general, the model should be tested (as a further work) in two directions, regarding *behaviors* and *economic*. For both these fields, more sophisticated and better defined AI-based behaviors (see intelligent agents specification like [fip97] or works like [ea02b]) and economic models (see, for instance, [AR06]) should be tested. From this perspective, these extensions should be tested directly on a SoRTGrid implementation (like those proposed in this thesis, realized on GridSim). The reason is that SoRTGrid framework grants the

possibility to test on an effective (logical or real) Grid, while SoRTSim is a simulator out-of-grid and produces scenarios that lack in the level of realism and complexity required for those advanced techniques and models.

Chapter 7

Conclusion

The presented work could be extended in different ways. Regarding SoRTSim, some improvements can be made in order to increase both the type of tests and the readability of results. In the first case, SoRTSim is currently being extended with the Multi-simulation feature. At present, SoRTSim allows to execute only a single simulation at once. The tests made in the different steps have required to execute similar simulations (i.e. that differ by a single parameter) in order to evaluate the impact of slight differences in the scenarios. All of this has been obtained by running a single simulation at a time, without the possibility to execute a set of simulations and to graph the differences among simulations of a single or a set of metrics. For this reason, the Multi-simulation module is under implementation.

Regarding SoRTGrid framework, the main future work is surely an extensive testing of the current implementation over GridSim. To this purpose, it is interesting to execute the same core scenarios used in SoRTSim tests. Comparing the SoRTGrid results with those obtained with SoRTSim could provide a measure of the efficiency of both SoRTGrid implementation and GridSim performance.

After this first phase, and, in case, proper modifications and bug-fixing of the implementation, SoRTGrid can be extended in three directions, regarding the *underlying Grid*, the *algorithms for the single components (User, Owner and Facilitator Agents)* and the *set of features*.

Regarding the first point, the current SoRTGrid implementation works on a simulated Grid over GridSim; some modifications of the interface toward the middleware can adapt SoRTGrid to work on other Grid middleware like GT4 (see Fig. 3.11 for further details). Such modification allows to test SoRTGrid on a emulated Grid (sets of Virtual Machines running GT4) or a real Grid (sets of physical machines running GT4) and to make comparisons on the test results.

The algorithms for the various components regard the definition and use of more advanced and complex routines at any side in order to test SoRTGrid in realistic and heterogeneous scenarios; for instance, effective learning-based algorithms at Owner side, different behaviors for User Agents based on AI and new remote discovery techniques on the Facilitator Agent overlay network.

The current features of SoRTGrid can be improved by adding support to more variables in SoRTBids and JRM, implementing a distributed and efficient support to detect abuses (e.g. overbooking) and a penalty system for the involved entities.

These kinds of extensions can take SoRTGrid framework to a very general and universal platform so that it can be used to test any model of time constraints or as a basis for the development of proper frameworks for supporting explicitly time constraints over Grid resources.

Bibliography

[AaAL+04] R. J. Al-ali, K. Amin, G. Von Laszewski, F. Omer, D. W. Walker, M. Hategan, and N. Zaluzec. Analysis and provision of qos for distributed grid applications. *Journal of Grid Computing*, 2, 2:163–182, 2004.

[AAAvL+] R. Al-Ali, K. Amin, G. von Laszeswski, O. Rana, and D. Walker. An ogsa-based quality of service framework. *Proc. of the Second International Workshop on Grid and Cooperative Computing (GCC2003), Shanghai, China, 2003.*

[AAAvL+03] R. Al-Ali, K. Amin, G. von Laszewski, O. Rana, and D. Walker. An ogsa-based quality of service framework. In *Proceedings of the 2nd International Workshop on Grid and Cooperative Computing, Shangai*, pages 529–540, 2003.

[AAHRW] R. Al-Ali, A. Hafid, O. Rana, and D. Walker. Qos adaptation in service-oriented grids. *Proc. of the 1st International Workshop on Middleware for Grid Computing (MGC2003) at ACM/IFIP/USENIX Middleware 2003. Rio de Janeiro, Brazil, June 2003.*

[AARW+02] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. G-qosm: Grid services discovery using qos properties. *Computing and Informatics*, 21, 4:363–382, 2002.

[AR06] J. Altmann and S. Routzounis. Economic modeling of grid services. In *Proceedings of the e-Challenges 2006*, October 2006.

[ARS98] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. An efficient k-means clustering algorithm. In *In Proceedings of IPPS/SPDP Workshop on High Performance Data Mining*, 1998.

[BAG01] R. Buyya, D. Abramson, and J. Giddy. An economy grid architecture for service-oriented grid computing. Technical Report 2001/84, 2001.

[BMA02] Rajkumar Buyya, Manzur Murshed, and David Abramson. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. In *Concurrency and Computation: Practice and Experience (CCPE*, pages 1175–1220. Wiley Press, 2002.

[BNTB06] P. Beckham, S. Nadella, N. Trebon, and I. Beschastnikh. Spruce: A system for supporting urgent high-performance computing. In *Proceedings of IFIP WoCo9 Conference*, 2006.

[CCD+05] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: a large scale and highly reconfigurable grid experimental testbed. Nov. 2005.

[CCM+06] A. Clematis, A. Corana, A. Merlo, V. Gianuzzi, and D. D'Agostino. Resource selection and application execution in a grid: A migration experience from gt2 to gt4. In *Lecture Notes in Computer Science n. 4276, 'On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE', R. Meersman, Z. Tari et al. (Eds.)*, pages 1132–1142, Springer, Berlin, 2006.

[CCT05] D. Talia C. Comito and P. Trunfio. Grid services: principles, implementations and use. *International Journal of Web and Grid Services*, 1, 1:48–68, 2005.

[CDC+] A. Corana, D. D'Agostino, A. Clematis, V. Gianuzzi, and A. Merlo. Grid services for 3d data analysis in virtual laboratories. In *Proc. 2nd Int. Workshop on Distributed Cooperative Laboratories: Instrumenting the Grid (INGRID 2007).*

[CFK+gh] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. Snap: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, 2002, Edinburgh.

[CGM+89] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, 1989.

[CLB06] A. Povzner C. Lin, T. Kaldewey and S. A. Brandt. Diverse soft real-time processing in an integrated system. *27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 369–378, 2006.

[DF] C. L. Dumitrescu and I. Foster. Gruber: A grid resource usage sla broker. In *Euro-Par 2005 Parallel Processing.*

[DHB+06] G. Deen, M. Hammer, J. Bethencourt, I. Eiron, J. Thomas, and J. H. Kaufman. Running quake ii on a grid. *IBM Syst. J.*, 45(1):21–44, 2006.

[Don08] J. Dongarra. *Urgent Computing: Exploring Supercomputing's New Role*, volume 4, 1. Cyberinfrastructure Technology Watch, P. Beckham, Ed., March 2008.

[ea02a] G. Fox et al. Grid services for earthquake science. *Concurrency and Computation: Practice and Experience*, 14, Issue 6-7:371–393, 2002.

[ea02b] J. Cao et al. Performance prediction technology for agent-based resource management in grid environments. In *16th International Parallel and Distributed Processing Symposium*, Washington, Brussels, Tokio, April 2002.

[ea05] D. Bernholdt et al. The earth system grid: Supporting the next generation of climate modeling research. In *Proceedings of the IEEE*, volume 23:3, pages 485–495, Shangai, China, March 2005.

[ea06] L. Malmstroem et al. Initial results of human proteome folding project on the world community grid: Structure and function predictions for biologists. 2006.

[Ear05] T. Earl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, 2005.

[ecl] Eclipse. *www.eclipse.org.*

[FBS07] W. Feng, D. Brandt, and D. Saha. A long-term study of a popular mmorpg. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 19–24, New York, NY, USA, 2007. ACM.

[fip97] Foundation for intelligent physical agents specification. *Available from http://www.fipa.org*, 1997.

[FK98] I. Foster and C. Kesselman. The globus project: A status report. Technical report, 1998.

[FK03] I. Foster and C. Kesselmann. *The Grid 2: Blueprint for a New Computing Infrastructure.* Elsevier, 2003.

[FKL+99] I. Foster, C. Kesselman, C. L., B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *In Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[Fos] I. Foster. What is the grid? a three point checklist. *Grid Today, July 22, 2002: Vol. 1 No. 6.*

[Fos06] I. Foster. Globus toolkit version 4: Software for service oriented systems. *Journal of Computer Science and Technology*, 21-4:513–520, July 2006.

[FTL+02] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 2002.

[gdt] Grid development tools (gdt). *http://mage.uni-marburg.de/trac/gdt.*

[GFGMI+08] S. Gorlatch, A. Ploss F. Glinka, J. Muller-Iden, R. Prodan, V. Nae, and T. Fahringer. A grid environment for real-time multiplayer online games. Technical Report CoreGRID - TR-0133, May 15, 2008.

[GJQ09] Jens Gustedt, Emmanuel Jeannot, and Martin Quinson. Experimental methodologies for large-scale systems: A survey. *Parallel Processing Letter*, 19(3):399–418, September 2009.

[GMCD08] V. Gianuzzi, A. Merlo, A. Clematis, and D. D'Agostino. Managing networks of mobile entities using the hyvonne p2p architecture. In *3PGIC Workshop at CISIS International Conference 2008 on Complex, Intelligent and Software Intensive Systems*, Barcelona, Spain, March 4-7, 2008.

[gra] The grace project. *http://www.gridbus.org/ecogrid/.*

[gri] Gridsim. *http://www.buyya.com/gridsim.*

[gxe] Gxemul. *http://gavare.se/gxemul.*

[IF06] A. Grimshaw I. Foster. The open grid service architecture, version 1.5. 2006.

[IFT01] C. Kesselman I. Foster and S. Tuecke. The anatomy of the grid: enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15, 3, 2001.

[JC02] S. Saini J. Cao, S. A. Jarvis. Arms: An agent-based resource management system for grid computing. *Scientific Programming*, 10-2:135–148, 2002.

[LCL07a] F. Meilai L. Chunlin and L. Layuan. Multiple qos modeling and algorithm in computational grid. *Journal of Systems Engineering and Electronics*, 18, 2:412–417, 2007.

[LCL07b] F. Meilai L. Chunlin and L. Layuan. Multiple qos modeling and algorithm in computational grid. *Journal of Systems Engineering and Electronics*, 18, 2:412–417, 2007.

[LK03] T.J. Lehman and J. H. Kaufman. Optimalgrid: Middleware for automatic deployment of distributed fem. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 164–171, 2003.

[mam] Multiple arcade machine emulator (mame). *http://mamedev.org/*.

[MC04] D. A. Manascé and E. Casalicchio. Quality of service aspects and metrics in grid computing. *Proceedings of Computer Measurement Group Conference*, 7th -10th February 2004.

[Men04] D. A. Menascé. Scaling the web: Mapping service-level agreements in distributed applications, September 2004.

[MJLM88] M. Livny M. J. Litzkov and M. W. Mutka. Condor: A hunter for idle workstations. In *Proceedings of the 8 th IEEE Inter-national Conference on Distributed Computing Systems*, pages 104–111, 1988.

[MSG05] J. Muller, R. Schwerdt, and S. Gorlatch. Dynamic service provisioning for multiplayer online games. *Advanced Parallel Processing Technologies*, 3756/2005:461–470, 2005.

[nee] The neesgrid website. *www.neesgrid.org*.

[NLN+05] K.C. Nainwal, J. Lakshmi, S.K. Nandy, R. Narayan, and K. Varadarajan. A framwork for qos adaptive grid meta scheduling. *Proc. of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, 2005.

[pla] Planet lab. *http:://www.planet-lab.org/*.

[RBG00] D. Abramson R. Buyya and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *Proc. of the Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, June 3-7 2000.

[RJAAW03] O. F. Rana R. J. Al-Ali, A. ShaikhAli and D. W. Walker. Supporting qos-based discovery in service-oriented grids. *Proc. of Parallel and Distributed Processing Symposium*, 2003.

[RRB] A. Harwood R. Ranjan and R. Buyya. Sla-based coordinated superscheduling scheme for computational grids. In *Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster 2006), IEEE Computer Society Press, September 27 - 30, 2006, Barcelona, Spain.*

[SCV+08] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and Buyya R. A toolkit for modelling and simulating data grids: an extension to gridsim. *Concurrency and Computation: Practice and Experience*, 20/13:1591–1609, 2008.

[sim] Simgrid. *http://simgrid.gforge.inria.fr.*

[WC04] J. Wen and G. Chang. Research on the financial information grid. *Grid and Cooperative Computing*, 3003/2004:698–701, 2004.

[Wel06] G.C. Wells. New and improved: Linda in java. *Science of Computer Programming*, 59, 1-2:82–96, 2006.

[wsr] The web services resource framework. *http://www.globus.org/wsrf/.*

[WWD08] Y. Wang, L. Wang, and G. Dai. Qgwengine: A qos-aware grid workflow engine. *Proc. of the 2008 IEEE Congress on Services*, 2008.

[WZS00] D. Olshefski W. Zhao and H Schulzrinne. Internet quality of service: an overview. Technical report, Columbia University, New York, USA, 2000.

[XDCC04] Huaxia Xia, H. Dail, H. Casanova, and A.A. Chien. The microgrid: using online simulation to predict application performance in diverse grid network environments. pages 52–61, June 2004.

[YC06] M. Ye and L. Cheng. System-performance modeling for massively multiplayer online role-playing games. *IBM Syst. J.*, 45(1):45–58, 2006.