EGEO:
An ENGINE in the Grid for the Earth Observation

V.M. Fulcoli, R. De Prisco, et alteris ex ESA-ESRIN
DISP, Uni. Tor Vergata, Roma; dip. Informatica Uni. Salerno, Fisciano, IT; ESA-Esrin Frascati, IT

Jan 16, 2006

Abstract

Today GRID is a sector of the IT research that covers a growing number of ideas and resources. It tries to exploit in the best way the synergy between distributed computers on the web using a set of protocols and services.

The gains in this sense are related to how simply can we reach compromises in order to achieve the optimal status between many aspects playing significant roles.

Scientific aspects (to elaborate a great data set in a dynamic context that needs data migration or replication and the subsequent program, fine-tuned modified, migration for the best site discovery) that I will analyze in order to obtain an increase in the calculus procedures have really been implemented in a system now operating at ESA. The system I studied have an evolution perspective pointing to integrate very different techniques, and my target will be to merge the common program with an intelligent GRID using 2 elements: The GridEngine and the jobFlow.

Objective for this task is the "intelligent Job" that is able to interact with others Job-entities of its same nature. The Job is the fundamental and the GridEngine is a meta-system added on Grid. The EGEO is a production environment in which this overlap is concretized trough the Grid onto Earth Observation.

KEYWORDS: GRID, GE-JOB, JOB FLOW, PROGRAM, GRIDIZATION

1 Introduction

Grid Computing [1] in the last period is becoming a paradigm for a new calculus generation: it allows the sharing and the merging of several heterogeneous resources spread and distributed among the www.net; supplying it with a real-time discovery of the best fitting computer or resource.

Grid is intended to allow the resources exploitation by users that can log on. The Grid concept arises immediately after the computer network; network services enable the simple communication, grid services offer a whole intercommunication environment for applications.

It is useful to notice that a Grid is a computer network, but a computer network is not a Grid.

The Grid components operate in several scenarios that sometimes replicate themselves with similar operating modes: the case of a process execution hosted by a remote machine. The Grid use typically has a Grid Interface that addresses the user in the different operations. Once the program is chosen it is uploaded on the Grid with execution's parameters specification and location of data to be analyzed. The Job so built is submitted to the grid and through the UserInterface it is possible to trace the log of the states reached by the Job during its execution.

A more evolved Grid vision leads to a growing in scenarios complexity. There can find place Brokers of resources or a pool of Brokers, Virtual Organization Management Systems (VOMS).

The GridEngine is the response to a problem focused in ESA [2] on the Mosaic-Image-Elaboartion. The Mosaic is a program that reading Meris [3] files operates a reference map (a sort of reticulation of the

[1] I. Foster, C. Kesselman, S. Tuecke, : "The Anatomy of the Grid - Enabled Scalable Virtual Organizations" , in "Supercomputer Application", 2001

[2] http://www.esa.int/esaCP/index.html

[3] http://www.esa.int/esaEO/SEMZ538X9DE_index_0.html

geographic region covered by the file), then elaborates images and on a correlation between spatial overlapped pieces of different images puts on the best fitting. Then collects the various tiles in one big result.

The crucial points of the solution to the Mosaic are:

1. An automatic gridization of the Job
2. The introduction of the minimal intrusion in the computational program

In the second point we used methods often adopted by parallel-computation, in the first instead there are techniques that you can find in the modern object oriented language programming.

An ulterior advantage raised up from this research: the GridEngine offers to computational programs the possibility for the exploitation of grid resources made synergic in a logic level more complex that enters in a Task flow organization logic. The capacity of traduction into a complex abstract language the whole operations chain, in which each operation is a gridEngine-Job gives a decrease in the whole difficulty for the final task in a distributed concept.

The GridEngine is an add-on for the Grid, it is applied onto the interface between Grid and whole NET. GridEngine raises the UserInterface intelligence allowing a quick insertion of a Job: an operation that is the main aspect for the research. The different Grids available now (as like projects or as implemented versions) either from the operational than from the architectural point of view were oriented towards the inner services (related to an increase of grid services functionality) and not towards the grid and the final user.

The computational software gridization is a complex operation that requires specific elements knowledge in order to be done. This is the crucial aspect of the research: a simplification for the gridization that does not decrease the total efficiency of the system.

On the basis of the Mosaic Project now in ESA it is being developed a plan of systematic exploitation of Grid resources and machines that operate in different fields: they are about 30 applications that use GridEngine, and its number is in a continous growing.

The fact I want to emphasize is that to obtain the "gridization" of the computational application you do not have to know deeply the Grid below. In fact once the whole system (composed by GridEngine and the various Grid-middleware and services) is correctly setted, the only constraints to be fulfilled are constituted by few simple rules: the whole complexity of the gridization is moved from here into a decoupled task that is performed only one time by the GE-Job builder.

2 The Grid and the status of the art

Grid is a model that takes advantage of many networked computers to model a virtual computer architecture that is able to distribute process execution across a parallel infrastructure or distributed infrastructure. Grid uses the resources of many separate computers connected by a network to solve large-scale computation problems either relating to a spread data set that follows from large data sets after breaking them down into many smaller ones, or providing the ability to perform many more computations at once than would be possible on a single computer, by modeling a parallel task sub-division between processes.

The Grid-computing is a conceptual framework rather than a physical resource. It is an approach that if utilized provides for the computational task with distant resources. The focus of Grid techniques is associated with the issues and requirements of flexible computational provisioning beyond the local domain.

An aspect that decouples Grid computing from distributed computing is the abstraction of the resource from distributed into Grid-aware. As result of this abstraction we can more easily accomplish a resource substitution. Some of the overhead associated with this flexibility is shifted in the middleware layer and in the temporal latency that has to be evaluated in terms of the impact on performances employing a Grid resource.

It's conceptual framework is evolving with a high rate, and the business side is involved in the commercialization, the science side is actively addressing the development environment and resource monitoring aspects. Activity is also addressed in providing grid versions for the High Performance Computing.

That environment is created to address the resource-usage that usually is characterized by its availability outside of the context of the local domain, and so an external provisioning approach creates a new administrative-domain: Virtual Organization with a distinct and separate set of administrative policies. The Grid job execution is distinguished by requirements created when operating outside of the local context, and a job is aiming to facilitate formalization and complying with the Grid context associated with the application execution.

There are now many projects related to the Grid technologies, characterized by several different aspects. If we consider as distinctive element the application field of the Grid we have the classification:

1. Computational Grid
2. Data Grid
3. Services Grid

If we focus our seeing in the manner the Grid is available to the final user then a classification could be, in a bottom-up style:

1. Middleware-Core
2. Middleware-User
3. Grid-Systems for Applications
4. Grid-Systems integrated

Among these projects, some concentrate on the development of software tools or services, others optimise the underlying network while others get ready for different scientific applications.

Each single project often covers more then one area of research, so a clustering of them could be:

1. Technology : involved in development of Grid-enabling technology
2. Testbed : oriented to developing and maintaining a production testbed with the existing Grids
3. Specific applications : oriented to exploration and harness of grid technology in some fields of science
4. Portals : Internet portals to grids, like grid on Demand

Technology

The *EGEE-Grid* Project endevours to develop the next generation of scientific exploration, task that requires intensive computation and analysis of shared large-scale databases, thousand of Terabytes, across distributed scientific communities. This project covers also the section related to testbed.

The *Globus Project* is developing fundamental technologies needed to build computational grids. It enables software applications to integrate functionalities, tools, computational and information resources that are managed by diverse organisations in widely distributed locations, including investigations for the security, the resource management, the communication protocols and the data management.

The *Biogrid* project strives to develop a series of analyzers on a network of supercomputers, allowing safe linkage and manipulation among various types of huge databases and facilitating systematic linkage among data processing requiring ultra high-sped computing resources, via ¡data grid technology¿ for the first and ¡computing grid technology¿ for the last one aspect.

Condor Project develops mechanisms that support High Throughput Computing on large collections of distributively computing resources, it is a workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queueing mechanism, scheduling, priority scheme, resource monitoring and management. Users submit their serial or parallel jobs to Condor that places them into a queue, chooses when and where to run the jobs based upon a policy, monitors their progress, and ultimately informs the user upon completion. Condor-G, a fully interoperable tools with resources managed by Globus, incorporates many of the emerging Grid computing methodologies and protocols.

The *CrossGrid* aims to develop services and programming tools for Grids in a large-scale domain, either with real-time simulations and visualisation in the fields of physics, earth sciences and medicine.

GRACE investigates for a distributed search and categorization that will produce an engine that enables just in time allocation of data and computational resources. It handles unstructured informations that are typically handled by search engines.

TestBed

The *DOE Science Grid*. It is building an advanced distributed computing infrastructure based on Grid, enabling the scalability in scientific computing.

AstroGrid builds up a working data-grid with associated data-mining facilities. A virtual observatory capable of supporting efficient exploitation of astronomical sets of data. Astrogrid will give a contribution to the Global Virtual Observatory.

GridPP. A collaboration of Physicists of high Energies and Computing Scientists who are building a Grid for Particle Physics. The main objective is to develop and deploy a large-scale Grid intended to be used by the worldwide particle physics community.

Teragrid is a project born to build a large, fast, and distributed infrastructure for open scientific research.The TeraGrid will include a great ensamble of computing power distributed at five sites, capable of managing and storing nearly 1 petabyte of data and toolkits for grid computing. These components need to be integrated and connected through a network that will operate at 40 gigabits/sec. .

Specific applications

There is *GriPhyN*, a project that merge information technology and experimental physics to provide the IT advances required to enable Petabyte-scale data intensive calculations.

PPDG has the purpose of enabling a distributed computing model for the high-energy and nuclear physics experiments. *PPDG* is actively participating in iVDGL together with GriPhyN for an approach to data grids for physics experiments.

The *iVDGL*. A Data Grid that give functionalities and tools to experiments for physics and astronomy. Share a computing, storage and networking resources across U.S., Europe, Asia and South America providing a unique laboratory.

Portals

GridOnDemand. A running project, largely operative in ESA, that allows the use of videly distributed resources, focused in a Data-Grid sector. it has a GateWay that serves as a preliminary logic package in order to collect informations useful to run Grid-Engine Jobs, as flows in a Task concept. It is thought for the earth-Observation applications bases itself on the inner layer constituted by the Grid-Engine Grid tool.

GRIDSTART has the goal of consolidating technical advances in the research of Grid-enabled applications. The www-site serves as a portal to a diverse collection of activities and initiatives taking place in Europ and in the United States.

3 A brief description

The field in which most Grid project express their power is the numerical computation: they points to the massive resources exploitation in order to minimize calculation latency. From a different point of view we can see the valence of Grid: to adjust the system in order to give the intelligence to the computational algorithm that will interoperate with the underlying grid middleware. It is a new entity, the GridEngine Job that is capable either of using the grid services than of interacting with its similar GridEngine Job, or maybe to recall from those some functions.

To exploit Grid services I added to the middleware (mostly of them a dataGrid like) on that User-Interface that is the gateway a new set of functionalities, embedded in a System and described by a protocol, to obtain an automatic insertion onto Grid of various applications tuned for the stand alone usage.

From previous paragraph you can see how the grids base their strength on their back side. I have never seen a particular attention devoted to:

1. give an ontological description of the application to be executed on the grid;
2. give to the Job its own identity and searching those conditions in which there are similar operational aspects;
3. facilitate the program introduction in the grid environment without intrusive operation on the code, only wrapping it in a new middleware optic, a facility that leads to the automatization of computational programs grid insertion;
4. allow to those programs, so wrapped, a cloning therapy to obtain a parallelism or symmetric-strong or symmetric-textual. The resulting application (the Grid Engine JOB that I will describe) could be cloned by using some uncostrained degrees of freedom, in such a way to make:
   • A strong symmetry, even starting from an internal tasks redistribution, and data to be processed;
   • A textual symmetry that relies to a different role of the program, respecting differnt costraints;

Those clones communicate between themselves through the GridEngine that acts as a "glue". All that obtained with no intrusion in the logical scheme of the algorithm.

This approach is a solution to a problem. In ESA researchers work on great data sets, obtained by satellite observation and spread on the Earth in several stations, with different archive systems. To elaborate these data we used a traditional approach: the single machine computation, serially. The data (known as products) have a level description: from P_liv_0 directly acquired by satellite sensor, to those affected by trasformations (for example geolocation operations give level 1 products P_liv_1 by particular algorithms ¡G¿), so we can express the serial computation on stand-alone machines as the one that elaborates data at each time, and the result on a set of k data (D) incoming has a total cost (¡S¿) sum of single costs

$$< S >_{tot} = < G > \sum_{i,(0-k)} D_i = \sum_{i,(0-k)} < S_i > \tag{1}$$

This cost is augmented because data are fragmented and must be collected onto the host machine for the computation, moreover the result cannot be stored on computing element but has to be moved on a storage. So, taking in account these factors, parametrized by α and β linearly to the data files, we have:

$$< S >_{tot} = < G > \sum_{i,(0-k)} D_i + \sum_{i,(0-k)} \alpha D_{i\_liv(n)} + \sum_{i,(0-k)} \beta D_{i\_liv(n+1)} \tag{2}$$

Now if the computer is only one it's clear that the cost is a waiting time, and the summation on single costs is a summation on single waiting-time per single-data:

$$< T >_{tot} = < G(t) > \sum_{i,(0-k)} D_i + \sum_{i,(0-k)} \alpha(t) D_{i\_liv(n)} + \sum_{i,(0-k)} \beta(t) D_{i\_liv(n+1)} \tag{3}$$

To reduce the total wait it is necessary to act on all 3 therms.

- For the first term
$$< G(t) > \sum_{i,(0-k)} D_i \tag{4}$$
we must operate in the direction of spread costs on several computers, so avoiding the serial elaboration and preferring the parallel one;
- For the 2 remaining terms the solution is contained in the preceding one: a multiple elaboration on those machine where the transfer cost were the lowest. So we reduce the total wait.

To this end we use the grid as middleware system that hosts the tasks, addressing them on available nodes (and minimizing the total cost). But now arise a problem: the treatment of first term need an

adjunctive cost related to the computational program transformation; from single-machine to grid-aware. That cost is not heavy but it assumes a deep user knowledge of the grid middleware, operating constraints and insertion procedures (often a class-ad or jdl job-definition-language).

Subsequently to that there is the problem related to data arising from a precedent elaboration:

$$G[D_{i\_livk}] \tag{5}$$

where

$$D_{i\_livk} \tag{6}$$

is a product from data produced by

$$G[D_{i\_liv(k-1)}]; \tag{7}$$

$$G[< G > D_{i\_liv(k-1)}]; \tag{8}$$

and with a multiple recurrence:

$$G[G[< G > D_{i\_liv(k-2)}]] - - > G^k[D_{i\_liv0}] \tag{9}$$

or, in the limiting but more frequent case, where the recurrency is not always made by same operators but made by an operator's class :

$$G[G..[H..[< I > D_{i\_liv(k-2)}]]] - - > G^j H^g I^l[D_{i\_liv0}] \tag{10}$$

where the operator is expressed by latin Uppercase (G) and relative costs by the ¡¿: ¡G¿.

Now we can see how the wait evaluation has become more complex in the reason that the cost is strongly related to the user's ability to retrieve, as soon as they are produced, the results of an intermediate computation; noticing that several operations could have common characteristics and the remaining could be completely different among themselves. This means that we must repeat the job description procedure for each computation, either in the case there are parameters unaltered and in the case they vary and are fixed by preceding operations. So, a procedure of the kind:

$$< G^\alpha > [< G^\beta > D_{generic}]; \tag{11}$$

implies a cost :

$$< G^\alpha >< G^\beta >=< G^\alpha > [D_i] + < G^\beta > [D_{i+l}] + < O > [D_i][D_{i+l}]; \tag{12}$$

where the last term in the second member takes in account the data D transfer cost, data D produced by G_{alpha} algorithm versus G_{beta}. So it appears how the optimal case is that in which cost factors for an O transfer are null. In the other way it get worse by intrinsic waiting times (related for example to the transfer throughput) and by human waits related to the user's quickness in the products transfer.

The goal of the GridEngine is to minimize these costs with the introduction of the Grid-Engine-Jobs that have their own autonomy and substitute the user in the data analysis description procedure, they moreover can tie together in a chain to implement the serial recursive elaboration. The result has a great effect. Only one job description and the GridEngine will build the whole set of G-E-Jobs for each application of that field. Moreover the GridEngine operates a linked execution of GEJobs monitoring each grid execution. The GridEngine is highly modular so we can adapt it to different grid middleware systems just with little modifications made on the lower layer.

4 The contribution of this research

A GEJobs manager (the GridEngine) and the EarthObservation as the application field, point to a system fully featured as the GridOnDemand web Portal. With this job we analyze technical advanced

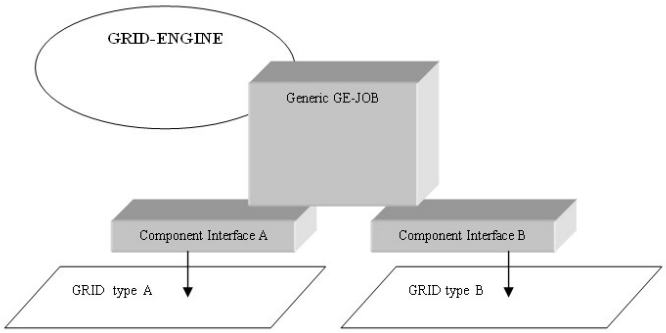

Figure 1: *The Modular structure of the GridEngine system*

solutions to hide the complexity of grids to scientists, having in mind that the focal point is to prepare a plan of development and integration for grid technologies that will preserve the computational applications' independency from grid-systems. The greater contribution that we will give in this work is the synergy optimization between applications and grids without modify the application itself, but giving to the grids an add-on (the GridEngine) that increase the complexity of the Interface GateWay between the grid and the external world (the POA:point of access). Then the GridEngine is able to manage operation flows (i.e. Tasks) built by GE-Jobs, linked in such a way to form joined distributions evolving to reach some particular and well defined conditions.

Each GE-Job can clone itself to migrate on the grid for a load balancing, and each clone can:

1. communicate with similar clones to exchange production informations;
2. invoke remote methods on different GE-Jobs to obtain informations (i.e. on the actual status or operation doing);
3. send new elaboration request (ex novo) to new GE-Jobs dynamically created on the fly;

After a long research of current grid projects and systems we noticed that never in them was been considered with particular attention the aspect less grid-aware: how an application can exploit the grid without suffer drastic invading operations at its structural level. For many projects the main feature is to offer inner services as distributed and functional as possible. For others the leading aspect is to reach the greatest grid-resources exploitation, but in these middleware systems it is developed a fine tuned compiling model for the computational program (i.e. the basic algorithm is broken down into pieces to be relinked again).

The GridEngine is a new service for the grids that is completely user oriented, acting as a generic grid service, in such a way to automate the complex grid insertion for applicative programs. It is a server listening on SOAP protocol for external requests (by users or remote programs as we will see later for the EGEO GridOnDemand) [1] and it manages GE-Jobs for splitting, cloning, inter-communication, remote call capability on alive GE-Job objects.

We investigated and realized a theoretical approach to the formal declaration of a GE-Job as an abstract class that could be instantiated (like the modern object oriented languages). The final scenario is a GE-Job Task that acts as real program that is executed on the grid; it has its main method from where new objects are created on grid, and each new object could be either a service provider or a GE-Job of the chain. So we have created a new meta-language for the grid, used by scientists to utilize in a transparent way the whole services grid-set. Further aspect is that the GridEngine is independent from underlying middleware.

With a 3 layers stratification we decoupled the GE-Job manager (residing at the upper layer) from the grid by the insertion of 2 intermediate layers, resembling the common concept of the ISO-OSI stack.

Inside a VO there are circumstances in which periodically a Job is executed without relevant changes, so we identify among these GE-Jobs a common unchanged parameter set as the intersection-set :

$$ZC = Job_k \cap Job_l \qquad \forall k, l \in \{GRID\} \qquad ZC \neq 0 \tag{13}$$

[1] EGEO GridOnDemand web Portal : http://giserver.esrin.esa.int/grid-dev/service/index.asp?

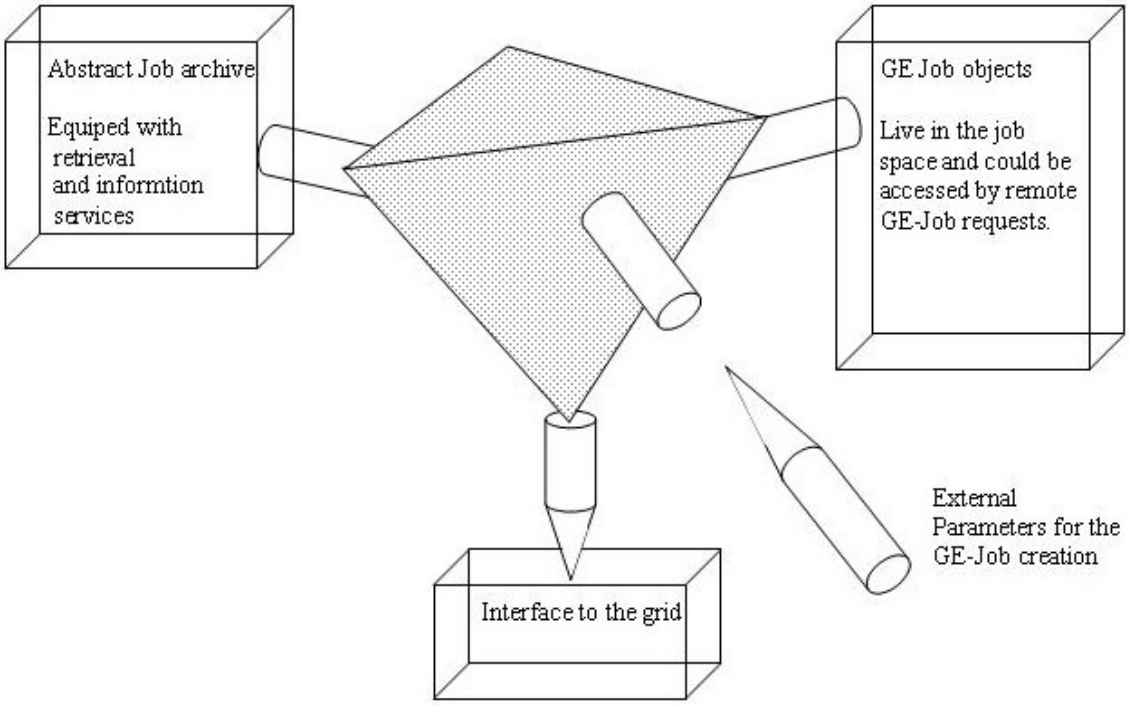

Figure 2: *The GridEngine and the GE-Job with the middleware and abstract classes*

This set identify a class : an abstract entity that once instanced becomes a GE-Job. The free parameters belong to 3 different levels:

1. strict-applicative acting directly on computational programs;
2. wide-applicative acting on the GEJob object;
3. generic acting at a middleware grid level.

When computational programs need to become grid-enabled we wrap it into a job_pilot program that drives its execution on the computing element and fixes some status advancing points useful for the GridEngine in order to control the job execution.

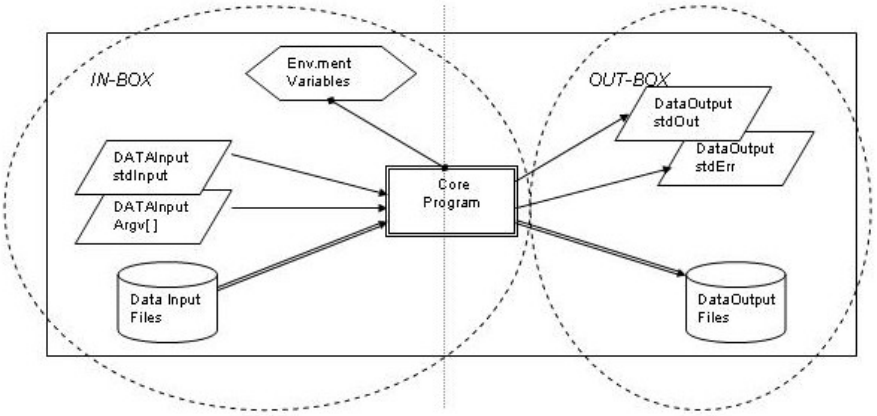

Figure 3: *The computational program in its InBox and OutBox abstract scenario*

The OutBox is composed by output and error process channels, by the OutputDATAset : the produced result set, OutputTable : a describing table for the OutputDATAset, with data description or metadata

or urls. The InBox is composed by input process channel, by the file Arguments resembling the argv[] parameters array, by the InputDATAset that comprises the whole data set to be processed, by InputTable that describes the InputDATAset as the preceding OutputTable, and the Environment hashTable that is a variable-value list for the environment parameters requested by process.

The GE-Job abstract class is built on the core-program, absolutely identified as regards to its architecture and linking factor to the stand alone host computer. The GE-Job class description is made through an XML file listing all object's methods and attributes shown to users, in a qualitative and quantitative description. A GE-Job has 4 basic statement parts:

1. Preparation: the phase in which all contextualizing operations and pre-conditions actions are developed;
2. Execution: (aka wrapper) effective execution of the created instance GE-Job on the grid middleware;
3. Completion: of the running processes and results collection followed by the run-time environment cleaning;
4. Status: describing the advancing point reached by running wrappers.

The XML must be of the form:

```
<?xml version="1.0" encoding="ISO-8859-1-GRID"?>
<!ELEMENT core - (info ?, config) >

<!ELEMENT info - (Author? & Context? & JobType? & Version ? & Schema? & Algorithm ? &
Description? &)>
<!ELEMENT Author - (#PCDATA)>
<!ELEMENT Context - (#PCDATA)>
<!ELEMENT JobType - (#PCDATA)>
<!ELEMENT Version - (#PCDATA)>
<!ELEMENT Schema - (#PCDATA)>
<!ELEMENT Algorithm - (#PCDATA)>
<!ELEMENT Description - (#PCDATA)>

<!ELEMENT config  - (Grid-Instance [ Local? & EDG? & Enea? & .... ]+)>
<!ELEMENT Grid-Instance - (Preparation? , Wrapper? , Completion? , Status?) [
Clean,stop,…... ]>
<!ELEMENT Preparation - (Static* & Template* & Virtual*)>
<!ELEMENT Wrapper     - (Static* & Template* & Virtual* & Pilot/)>
<!ELEMENT Completion  - (Static* & Template* & Virtual*)>
<!ELEMENT Status      - (Static* & Template* & Virtual*)>
<!ELEMENT Static  - (name)>
<!ELEMENT Template  - (name , run-as)>
<!ATTLIST Template  type
        Normal
        Exec
        Method
        Pilot  #REQUIRED>
<!ATTLIST Template  archive
        yes
        no #REQUIRED >
<!ELEMENT name - (#PCDATA)>
<!ELEMENT run-as - ( description , env )>
<!ELEMENT description  - ( #PCDATA )>
<!ELEMENT env  - ( #SysINFO )>
<!ELEMENT Virtual - (name)>
```

Figure 4: *The XML description for the GE-Job config.*

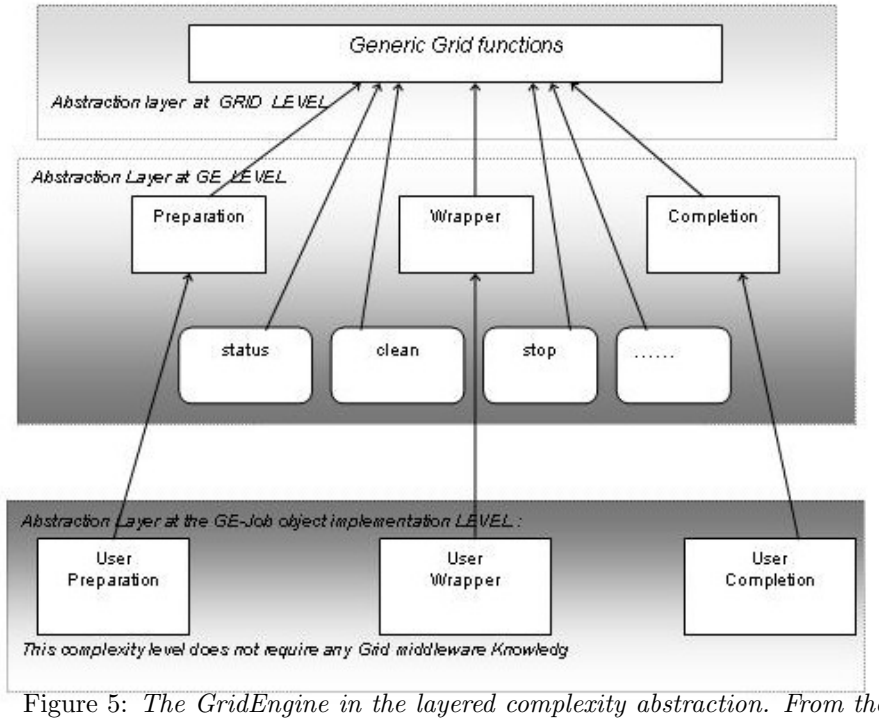

Figure 5: *The GridEngine in the layered complexity abstraction. From the middleware to the user in a top-down view.*

Each component of the GE-Job object in the abstract class representation is really a template file that is implemented and instantiated by the GridEngine during the init step. We define 3 template kinds:

1. Static : it's part of the Input archive anyway, and can be a pilot; it is not copied in the GE-Job Local working Directory but is directly uploaded on the grid during preparation phase;
2. Template : the generic skeleton file to be defined; it is instantiated and copied from Template directory to Working Directory. Normal is only instantiated, an Exec is also executed in the wDir work-space. Method is callable by remote requests on the GE-Job object.
3. Virtual : not appearing in the GE-Job class repository, but created on the fly by local operations.

The pilot program is part of the archive to be uploaded and is the driving wrapper program to the computational application that must be run remotely. Notice that the word Remote (or Remotely) stands for operations to be executed on the grid computing element, while Local (or Locally) stands for operations made on the grid gateWay machine.

At this point the similitude between GE-Job and Object Oriented language's (like Java) object is clear. We can create a GE-Job object and invoke methods on it. It becomes part of a greater schema: the Task, in which we collect many GE-Object in a chain of operation, controlled by the GridEngine in their evolution. This is the EGEO meta language with whom it is possible to build many job flows in which each atomic piece makes the pre-conditions for the subsequent (there could be one or more) GE-Job.

From this point of view the GridEngine is a meta-service for the Grid. A **service** is an entity able to accomplish specific tasks. A **web service** define methods and technics in a key distributed schema paradigm: representing software components visible by users, showing the way users can utilize them and how to identify these functionalities. A **grid Service** is a web service that offers a well defined interface set which help in the identification, dynamic creation, life monitoring, event notification and monitoring; its protocol provides scalability and nomenclature. The **Meta service** here offered by the GridEngine is a grid Service with the difference that this last works on system and grid resources while it works on GE-Jobs, that in their turn operate on system and grid resources.

5 The GridEngine and the GE-Job

The GridEngine is basically a GE-Job manager. It has 4 principal blocks: the GE-Job and session managers, the system manager and security controller.

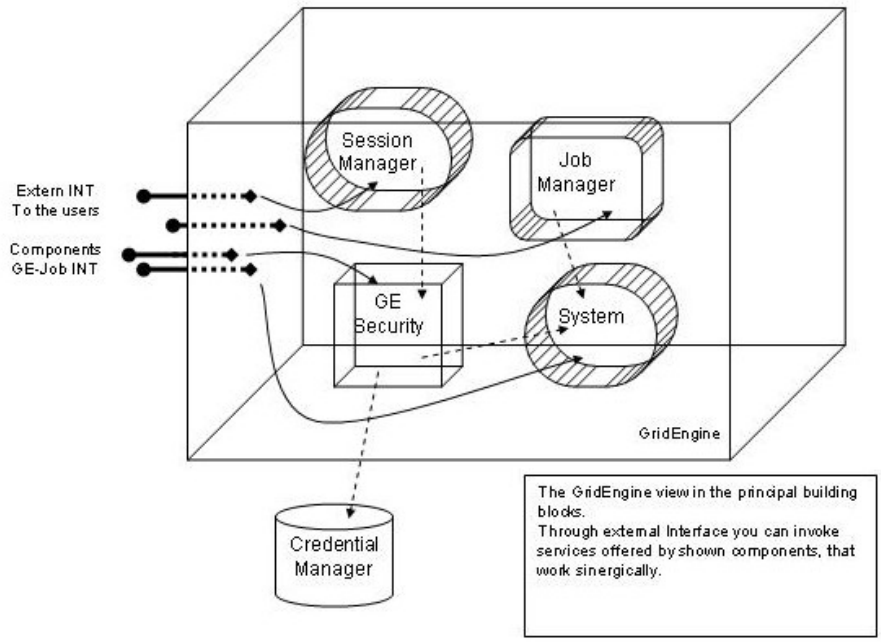

Figure 6: *The GridEngine and the principal components.*

To describe the GridEngine we analyze a real case: the remote elaboration request for a GE-Job. The user could be either an external client (i.e. the GridSurfer we made that is optimized for the GE-Job service test), or an external service that uses the GE services (in fact the GridEngine offers a GateWay to the grid not only for the GE-Jobs communication, it could monitor several middleware aspects), or a GE-Job itself that invoke GridEngine's services during its operational phase. The first step is the client's identification through credentials. Then the client creates a work session, managed by session manager, that stores all client credentials and all the GE-Job objects that the client arranges in the operation plan (i.e. the so called progamming meta language). Then a GE-Job, referring to a particular session, correctly instantiated is in its initial state and could be executed if the preparation step ends successfully. The Job's submission via the GE launches the computational process on the computing element and the job_status informs the client on its progress and middleware status of the pilot. To the execution's end follows the job completion. All these operation can be launched one by one by client or automatically by the GridEngine, that starts the subsequent operation when the preceding and propaedeutic one is reached. All the informations related to these particular status progresses are called checkPoints and are fixed in the GE-Job definition: each running GE-Job advances in its task and informs the GridEngine when reaches its next checkPoint.

This double mechanism is useful when the client needs to test the various GE-Job, so he can monitor step-by-step, and when he then uses the GE-Job as operative. Besides Preparation, Wrapper,Complete and status, there are also the :

1. Stop, acting first on the middleware process and backward to the Local GE-Job process;
2. Clean. As the prededing one it operates backward cleaning all the GE-Job's files and informations.
3. Job_Get_Up, tries to stop the current GE-Job and a following restart from the last reached checkPoint.

The checkPoint is a point in the state-space for the GE-Job, and represents a well defined stage from where it is possible to restart the task.

The following schema illustrates a possible status flow for a GE-Job in its state space.

A set of GE-Job in a session is viewed as a task: it simulates a Chain in a job-Flow driven by the GE meta-language. The GE-jobManager has a complete vision of the whole session's GE-Jobs state-space and operates the further preparations when dependency constrains are matched.

The figure shows a typical GE-Job linking in a Task. The $GE-Job_\alpha$ has no dependencies so could be Prepared immediately after the creation. The $GE-Job_\beta$ depends on $GE-Job_\alpha$ and its dependency

Table 1: The checkPoint Table: Evolution of states between well defined stages.

CheckPoint	Description	value
1	JOB_STATUS_CODE_UNDEFINED	0
2	JOB_STATUS_CODE_INITIALIZED	2^0
3	JOB_STATUS_CODE_REJECTED	2^1
4	JOB_STATUS_CODE_PREPARING	2^2
5	JOB_STATUS_CODE_PREPARED	2^3
6	JOB_STATUS_CODE_REFUSED	2^4
7	JOB_STATUS_CODE_SUBMITTED	2^5
8	JOB_STATUS_CODE_TERMINATED	2^6
9	JOB_STATUS_CODE_ABORTED	2^7
10	JOB_STATUS_CODE_COMPLETING	2^8
11	JOB_STATUS_CODE_COMPLETED	2^9
12	JOB_STATUS_CODE_CORRUPTED	2^{10}
13	JOB_STATUS_CODE_CLEANED	2^{11}

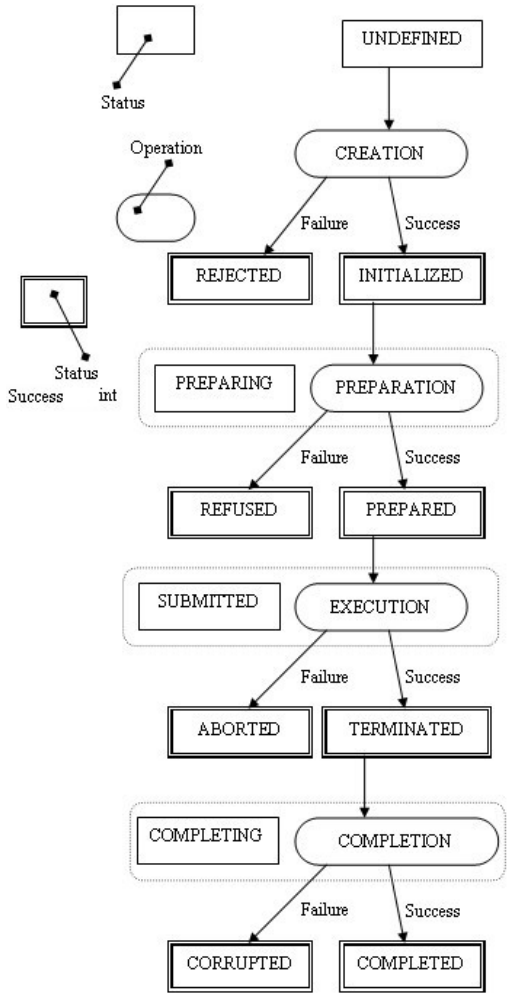

Figure 7: *The GE-Job checkPoint sequence.*

is related to reaching of COMPLETED status for $\mathrm{GE-Job}_\alpha$. It is the GE-Job that informs the

GridEngine on the reached status. The states of $GE-Job_{\alpha}$ are traced on the state-space and the same for $GE-Job_{\beta}$ till the achievement of next COMPLETED.

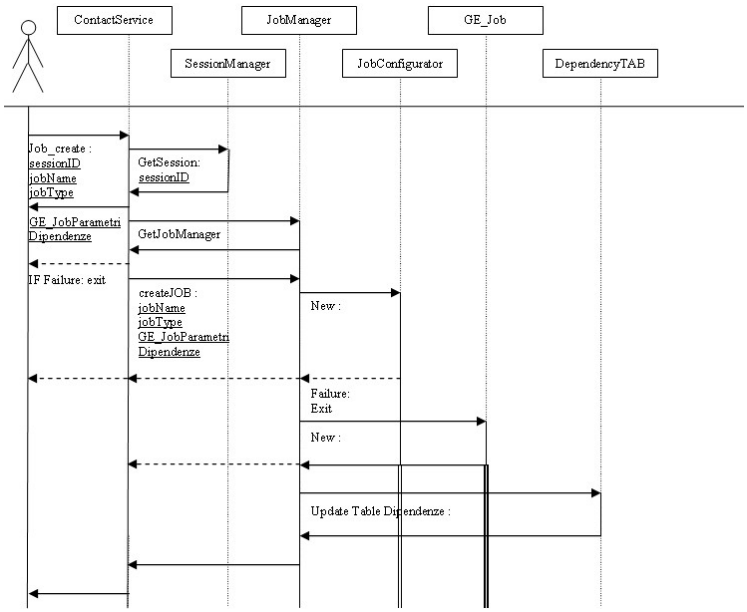

Figure 8: *The GE-Job sequence in a Flow diagram.*

Each GE-Job can be constrained by relations different from the COMPLETED achievement by its binding one: for example you can bind the Preparation of the $GE-Job_{\psi}$ by the PREPARED for the $GE-Job_{\phi}$ and by the TERMINATED for the $GE-Job_{\chi}$. In this case when all the conditions of binding are meet the next operation could take place. So we can define more typology of dependencies:

- Unitary, when a GE-Job depends on one different GE-Job;
- Multiple, when a GE-Job depends on more GE-Jobs;
- Simple, if the binding constraint is related to the correct completion of the preceding GE-Job;
- Partial, if the binding constraint is related to the reach of any of checkPoints for the preceding GE-Job;

6 The GE_Job_splitting and the remote method invocation

To submit more than only one GE-Job you have to create all those you need and then submit them, if these GE-Job are object of the same class you can use the splitting that optimize the memory and

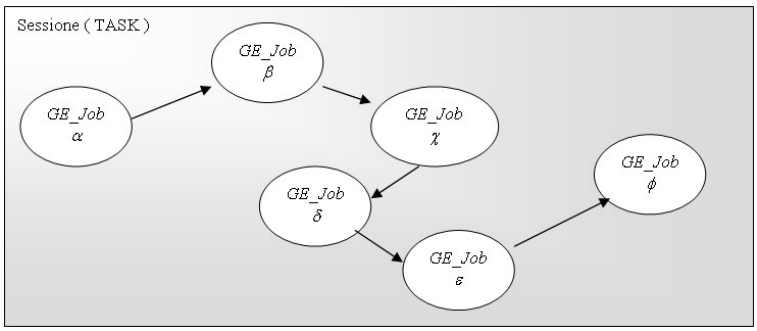

Figure 9: *The GE-Job chain sequence in the session Task. This is the case in which each GE-Job has only one binding ancestor.*

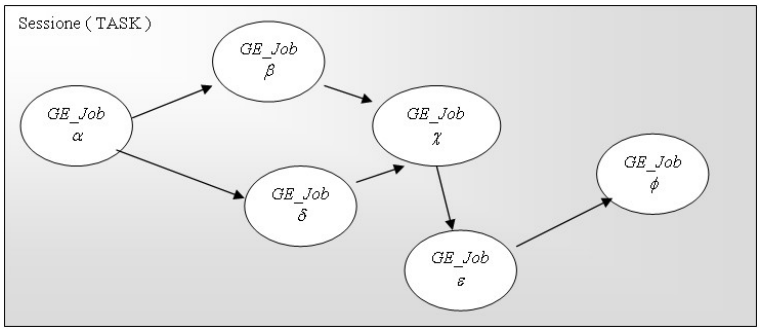

Figure 10: *The GE-Job chain sequence in the session Task. In this case it is shown a case in which a GE-Job has more than one dependency.*

resources leak. After the splitting of a splittable GE-Job we have several images (clones) of the source object, each one is different upon variations of the sensible part. A GE-Job is splittable if it is possible to istantiate more than only one GE-Job from a single source. Each clone has a common part with the source GE-Job (the static part) and is different upon the sensible part.

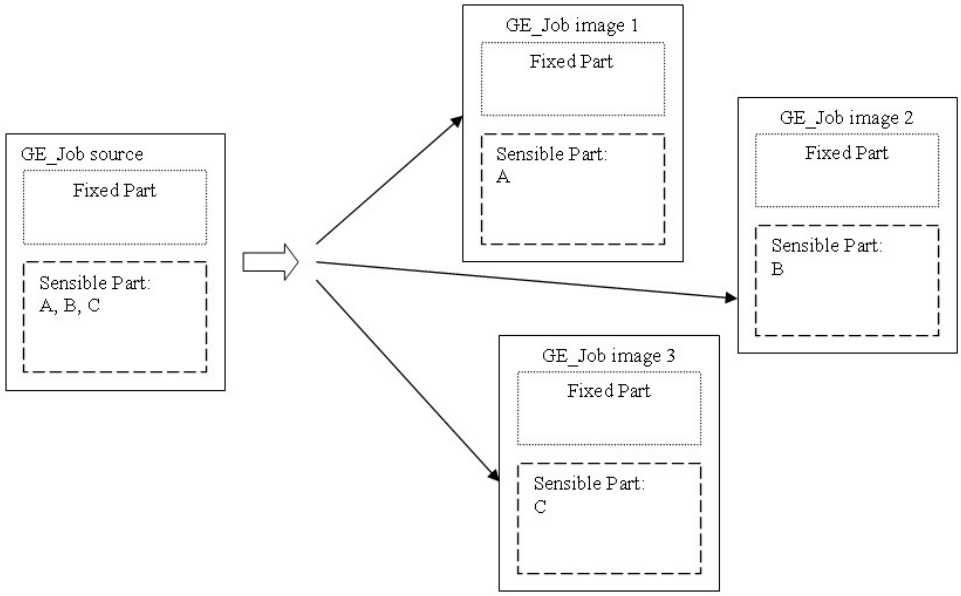

Figure 11: *The GE-Job chain sequence in the session Task. This is the case in which each GE-Job has only one binding ancestor.*

The GE-Job splitting is the following: we are going to analyze a GE-Job source composed by a sensible part S for k GE-Job clones and m sub-elements. The sensible part is a set of elements S_j. We will have for each sensible part m elements as combination among all the elements of S_j, with or without repetitions as the GE-Job requires (the number m then could increase depending on the repetitions).

This mechanism is useful in the simple parallelization of GE-Jobs on the grid. The simple parallelization arises when on CE's we run k similar processes, clones with different IN-Boxes. Each clone receives as input data to process a sub-set of the starting Data set. In the literature there are studies of performances related to this parallelization scale, and for R.David, S.Genaud, A.Giersch, B.Schwarz, E.Violard "Source code transformations strategies to load-balance Grid applications" in the master-slave case we have the minimal execution wait when each node has an input data that is inverse proportional to its computing power. In a different approach we can use the belongings logic scale, it describes levels to which an entity could belong and with whom it can communicate.

At the Task level we see entities communicate through a white board, shared-memory, that allows tasks to post messages and data. This communication is between GE-Jobs of the same task or differen one; these values have a global scope and can be viewed by all tasks Jobs. At the GE-Job level we

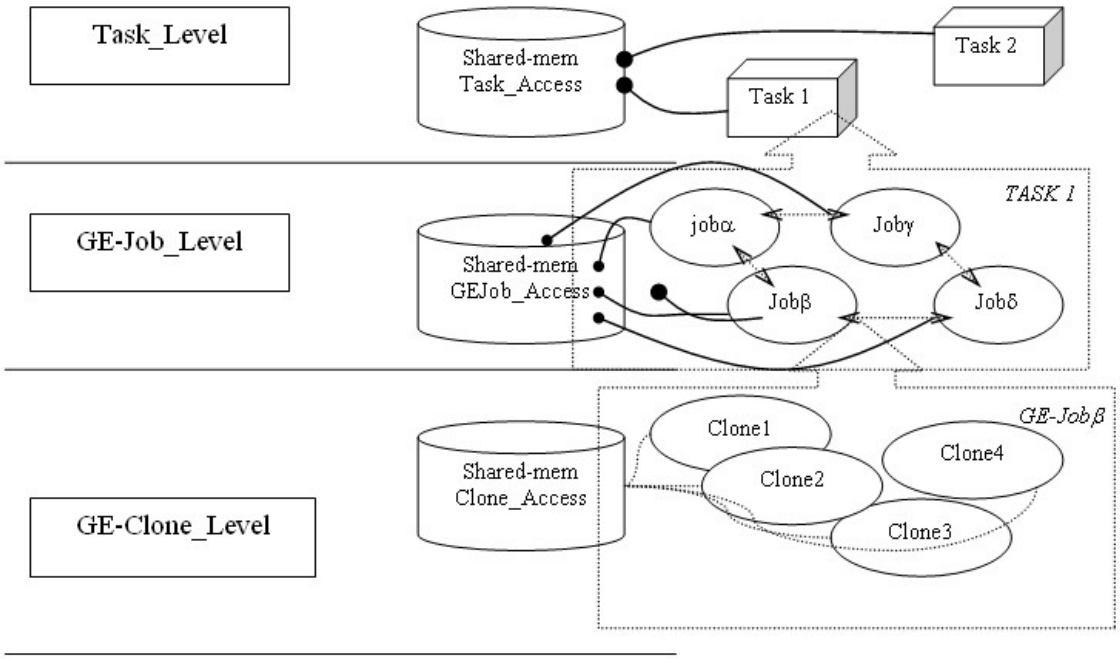

Figure 12: *The logic schema in communication and belonging for clones and GE-Jobs.*

see of two kinds of communication: or on a white-board or through point-to-point channel between GE-Jobs. The allocated shared-memory for the GE-Job level is accessible only for those GE-Jobs belonging to the Task, so the visibility is restricted to the Task. A GE-Job posts messages or data on the white-board for the Job level to spread unaddressed information related to its state. If it needs to exchange sensible data about a defined GE-Job it uses the point-to-point channel that involves only a 2 objects communication.

At the lower level, clone level, we have a communication with a LIFO white-board. The clone GE-Jobs have visibility either for higher levels than for their own level in which intercommunication is restricted to only those clones belonging to. The remote GE-Job method invocation is accomplished at the GE-Job level. A meta program in the GE-Job logic language has the main GE-Job object from which starts the whole elaboration that goes through functions performed either by itself than by other objects. The meta program uses a general view at the creation time: in that view it is related to the task a time-causal evolution. Each GE-Job of the task is a child of one or more parents and is parent of one or more children. During its life a GE-Job can:

- create,
- execute a new GE-Job object in the GridEngine,
- then call on it a particular service,
- and then terminate it.

In a grid this is always possible, and there is a great difference with static services, like web-services. The services offered by GE-Jobs are dynamic as regarding the resource exploitation, they can migrate to different sites if the CE is overloaded. If web-services, as static services, have a responsivity dependent upon their number (fixed) and upon host machines performance, the GE-Job respond to a request grow with a re-creation in the grid, till satisfaction.

7 A comparison of more competing technologies

To evaluate the project exposed (and the real tool yet built and operating at ESA in the EGEO environment) is useful to make some comparisons between similar technologies, maybe belonging to the same applicableness class. With a performance comparison we see clearly that we eliminate waiting times due to repetitions of cyclic operations. A comparison with the gateWay-interface of the grids, for example the egee User Interface [1] shows that the GridEngine is however more advanced why it is built on egee-UI services and offers more refined and complex functionalities. Seeing the grid activities panorama a correct comparison with this automation Job system (GridEngine and EGEO) is unrealizable because this dual entity does not exist. So practical and formal evaluations are made by measuring total elaboration waits, and a reduction of a factor 5 or 6 for the costs, operating with a pool of low power computers (7 Dell OptiPex 150 + 1 Beowulf) respect one SiliconGraphics (SGI Altix 4700), due of course to a different resource exploitation approach. A system that could be a comparable term, not comprising all the characteristics, is the condor DAG-MAN [2]. It has some aspects resembling the GridEngine as regard the work sessions and Job-Flow. I want to emphasize that our research was developed at the same time with DagMan. The DAG (Directed Acyclic Graph) manager is responsible for the sequential execution of programs that need to be submitted to Condor. It needs the list of programs to submit and the list of pre and post executions, a dependency description and retry number for failure. This system is similar to the TaskFlow contained in the GridEngine, but the GridEngine has different functionalities. The DagMan essentially is a dependency manager, the GE is a gridization programs tool, that then monitors dependencies. DagMan furnishes a pre and a post program for each Job. These auxiliary programs are added to the core program in order to be run before and after the core execution. The core program is submitted to condor when constraints are matched, as checked by dagMan, i.e. is executed the pre-program, then with a successful exit status the post-program. The GridEngine has a dual with its Preparation and Completion that surround the wrapper (Execution). In the GridEngine there is a further specialization concerning the start of children GE-Job's execution at well defined intermediate checkPoints reaching. This is for assuring system a flow efficiency and inter-communication on 3 levels. It is the 2^{nd} level communication that allows progress in subsequent steps that can be more or less grained, grain that allows a GE-Job in the task to start running not only at reaching by its parent (parents) of successful completion but at reaching of some pre-defined checkPoints. This procedure has proved to be very useful in dealing with Mosaic [2] Job project at ESA, in which clones can launch subsequent operations before they terminate the whole data computation. In fact it could happen that a first data chunk if correctly elaborated is ready for the next GE-Job, that for the mosaic operate an inter-correlation on available chunks. And it is useful also on fault cases, during processing, when it does not preclude the task advance even if it's available only a portion of expected data, as intermediate elaboration result on not constraining data (see fig. 13).

8 A real scenario for the GridEngine and future outlook

The EGEO in ESA is now an operational system, composed by:

- several grid middlewares,
- the GridEngine as GE-Job-Manager,
- a web-portal that introduces friendly the users onto grids having all capabilities to retrieve metadata for data analysis.

EGEO is fine tuned for computation in the Earth Observation field and is capable of increase computational power expressed by research applications. The main feature for EGEO is constituted by its cache-like structure that support a continuous data processing. Often the not-functional features of distributed applications, when measured and evaluated, are rather far away from those expected because variables concerning the problem are many and rapidly varying. The presence of data-nodes

[1] http://egee.cesnet.cz/en/voce/ui.html
[2] http://www.cs.wisc.edu/condor/dagman/
[2] This Mosaic is a task made by 3 kind of GE-Job, as we will describe later.

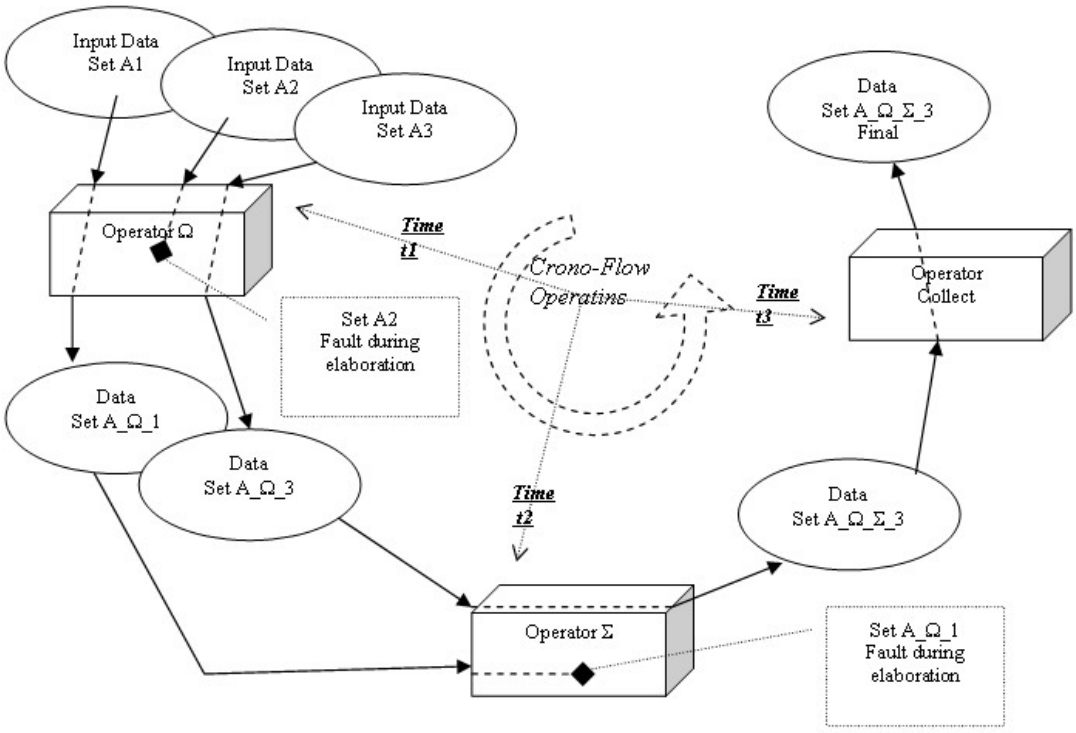

Figure 13: *The advancing in steps during a task (Mosaic) viewed from a time point of view.*

and computational-nodes lead to the arrangement search between data migration toward processes or vice versa, in an intermediate point. To find the optimum between these extremal points is often impossible a priori, so we need an investigation of the particular case in a run time way basing on several parameters. For the connected-parallel and fully-parallel applications it is possible to reach the minimal cost simply growing the number of CEs, in a theoretical approach it could be possible to reach the lowest cost of only data transmissions on the network, but in practice the lower level is reached before than that one predicted as a data transit cost.

The principal researchers need is the access to data, algorithm and distributed resources; the present paper illustrates the research in that direction. Each application for the Earth Observation field operates on a well defined data set: satellite and acquisition sensor, time and spatial interval of the acquisition, orbit number, refinement operations type. With the exposed characteristics we identify data set required by the computational algorithm, and the program itself has a precise identity when regarded in the space of applications for the EO field.

- Each GE-Job is accessible to researchers, and it is identifiable by a parameter's set;
- The products nomenclature that are available for users having rights.

EGEO in this sense does not only control data and algorithm distribution but manages the belonging to users, or groups, or Virtual Organizations. In the light of what I said the gain is immediate for the embarassing parallel applications in which a simple data redistribution on copies (of the original program) running on free remote CEs resembles the Clones splitting.

When the user requests some data set elaboration through a computational algorithm, using the EGEO, he starts a complex operation that begins with the request on a WEB GateWay interface: the GridOnDemand web server. It manages the first step of the sequence chain of operations, having the logic of traduction into data files set the time-spatial region covered by user. Then it translates into a meta-description language,for the GridEngine, the way the user requests for those data elaboration and sends it to the GridEngine. The GridEngine subdivides tasks basing on data to be processed:

- if they exist,

- if they are available,

and does it checking on the Persistent-Catalog. Data found can be utilized by GE-Job through a collection executed by GridEngine. The above pre task collection is a general purpose GE-Job: Collect, that moves on the grid storage elements data found and then marks them (either with operational than ontological attributes) as belonging to the particular session and defined GE-Job. When the GE-Job is forwarded to the grid (if there is not a grid Resource Broker the GridEngine will manage partially this feature) it becomes really a series of processes on computing elements; afterwards each result (either for a partial result than for a final result) is registered on the Cache-Catalog. At the chain's end the GridEngine launches the GE-Job Publish: the final recollector of the whole result set produced by intermediate GE-Jobs, that evaluates the Ge-Jobs successfully termination and

- moves products in the OUT-Data place for the final recovery step made by the user, or
- cancels and cleans the EGEO grid Cache.

The EGEO has several access points: one of these is the GridEngine Front-End.

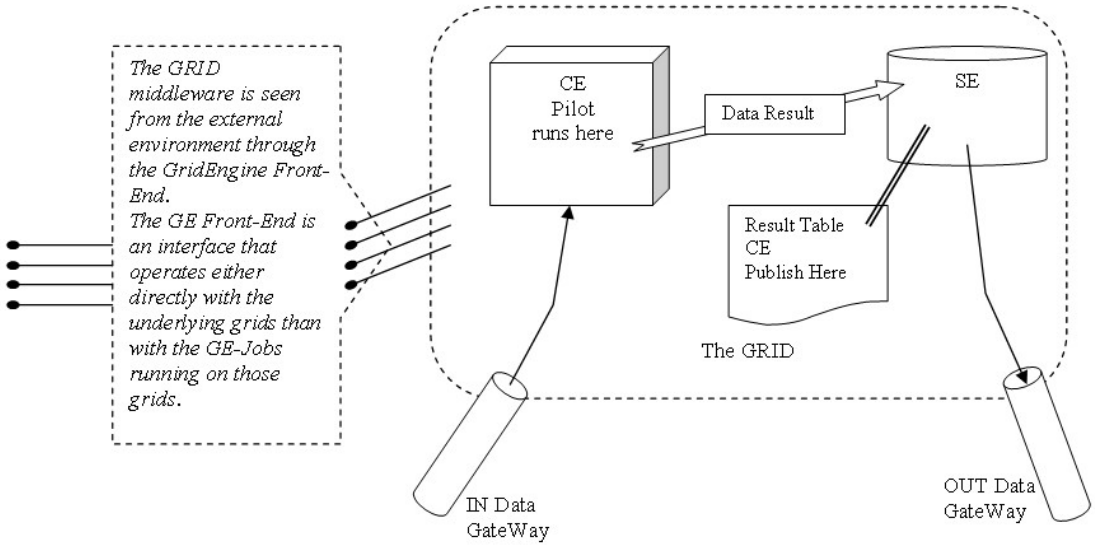

Figure 14: *The EGEO as appears when viewed from the net.*

As an example of using the EGEO we analyze the Mosaic project. It is a task made by 3 different sections:

1. Structure;
2. Tessera-Elaboration;
3. Recompilation;

The 1^{st} and the 2^{nd} piece can be splitted on sensible parameters:

- Structure[job$^{[k]}$: k $\in (1; ...L_1)$] where the upper limit L_1 is given by graining.
- Tessera $-$ Elaboration[job$^{[k]}$: k $\in (1; ...L_2)$] where the upper limit L_2 is given by row data files number.

On the Mosaic-Project there are 2 further GE-jobs, one of them is the GE-Job Collect that is a general utility Job placed as first step of the chain. It collects files requested by user querying External

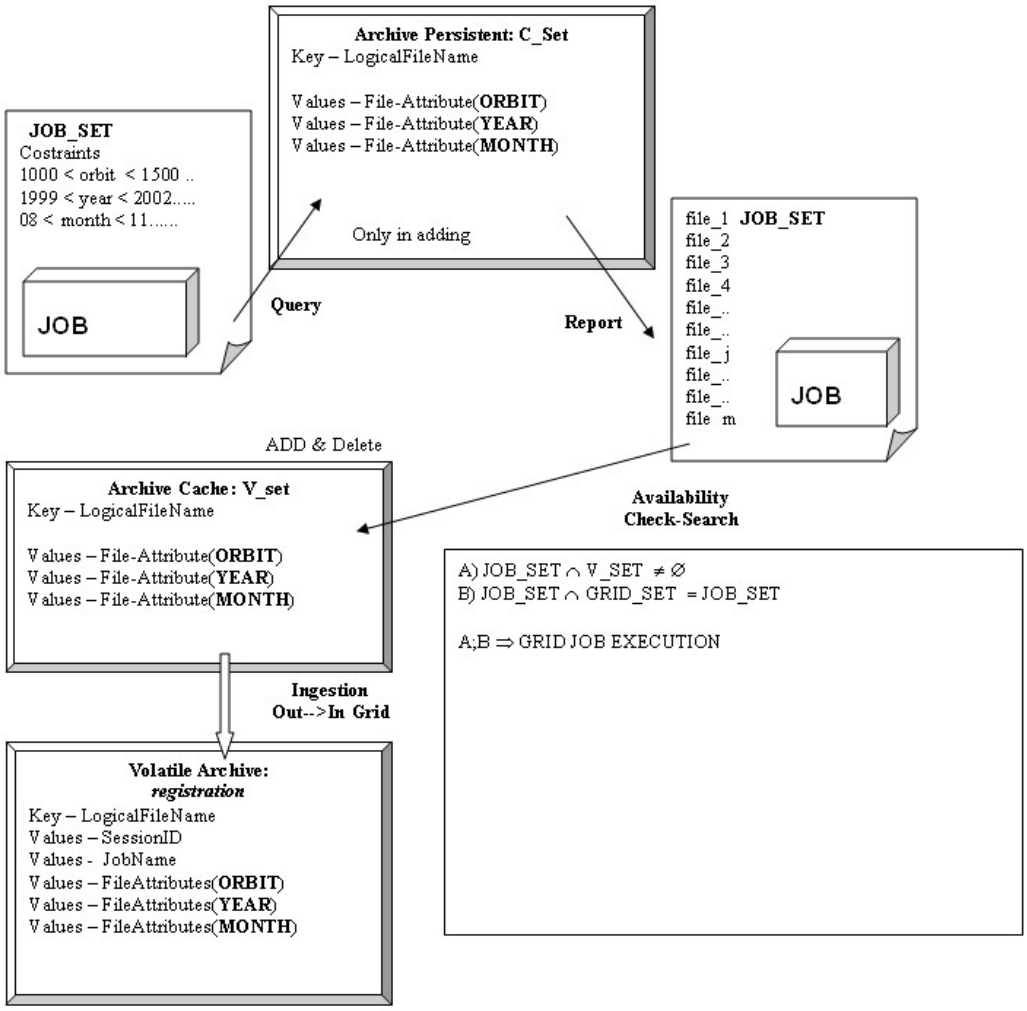

Figure 15: *The Collect GE-Job during the meta information traduction phase.*

Persistent Catalog and registers the correct upload on grid storages. The OutputTable is registered as output-result for this Collect Job, with attributes that identify sessionID and Name.

The Mosaic beginning part,i.e. the Structure, parses some request-parameters (the space part Interval : the Bounding Box) in order to prepare the Mosaico-grained-Grid where to place the further tessera.

This step is a global geo-location. Then Structure parses the Collect's Output Table (containg files meta data), to identify on Cache Catalog if there are copies already processed for any region covered by Space-Time interval of the present Mosaic. The central part of Mosaic project (the analysis and tessera elaboration) follows the initial stage, the Structure (i.e. structural data creation either for the geo-localization of input row data than for the geo-localization of result data).

Both pieces of Mosaic project (the Structure and the Tessera elaboration) can be composed by many collaborating GE-Jobs: clones.

The communication intra-clones is managed by the GE-Job white-board that simulates a parallel process communication, it is so specific for the computational application. The clone-clone communication is offered by the GridEngine.

In the picture 21 are shown 2 Structure GE-Jobs [StructureA and StructureB] that are distinct computational applications and not clones. They operate in conjunction and need to exchange data between themselves, because they do not stand at the logic level occupied by clones, they communicate through the Session White-Board service offered by EGEO Volatile Catalog. It is possible then to think of distributed applications interacting between themselves. Useful for that logic level is the GE-Job invo-

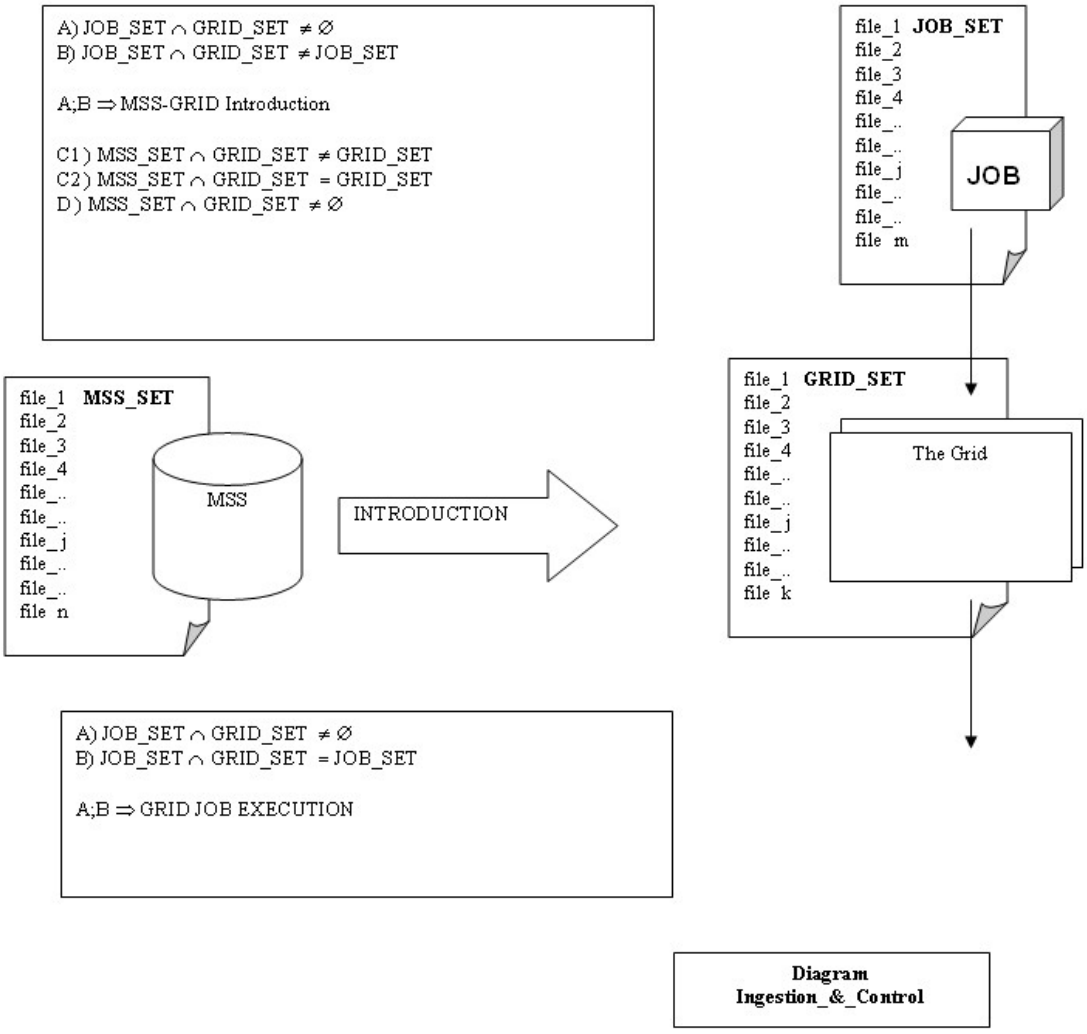

Figure 16: *. The MSS stands for Mass System Storage, generally constituted by batteries of tapes driven by robots.*

cation by another GE-Job: the Session White-Board stores meta data and on a GE-Job it is possible to request the execution of a procedure that it offers.

When a GE-Job becomes an object, during the instantiation, it reaches its own space and operates either for number crunching than for information dispatcher; so it arises a dynamic splitting of row data. During the initial stage all files are found and the clones migrate onto Computational Resource following a theoretical prediction, and then is the GE-Job that refines that prediction in a corrected prosecution having checked the files that are nearest [here I mean nearest regarding the position in the network space furnished of an inner metric: a metric normalized and inverse proportional to the throughput] and have a less operating cost; and the GE-Job informs the Job White-Board about its new set.

Each clone runs till $clone_{j-th}$ has interest to know if data-file to be computed ($file_{k-th}$) is scheduled by some other clone, if not it goes to elaborate.

Informs the GE-Job White-Board that the $file_{k-th}$ is being processed by $clone_{j-th}$; but if there happens a fault or an exception, it stops the current execution and asks for the prosecution another GE-Job clone not heavily loaded: so calls the remote method on that $clone_{y-th}$ to control the load. A low load response will followed by the sending of delegation request for elaboration. The Mosaic Project is fault tolerant in the EGEO system.

Now in ESA are available in the EGEO environment (aka GridOnDemand when integrated with the web Portal) several GE-Job offered by scientific community by a team working at ESA, and in which I am involved.

The number of scientific applications that as GE-Jobs utilize the EGEO system confirms the efficiency of GridEngine.

The whole system has as a FrontEnd constituted by a web GateWay portal, hosting functionalities offered by GridEngine, by underlying Grids, by Data Catalogues, by Mass Archive storage systems. Through the portal the users can execute the Tasks listed in the Table 2, moreover it offers several services made by the conjunct utilization of modern GeoInformation techniques and remote web services.

The picture shows the first stage of operation chain. A Task where 4 GE-Jobs are linked to build an operation-flow.

The picture shows the instantiation phase, automatized for different GE-Jobs making the Task.

In the figure we can see how the GE-Job number 1 of the chain is running and the subsequent are waiting for its completion.

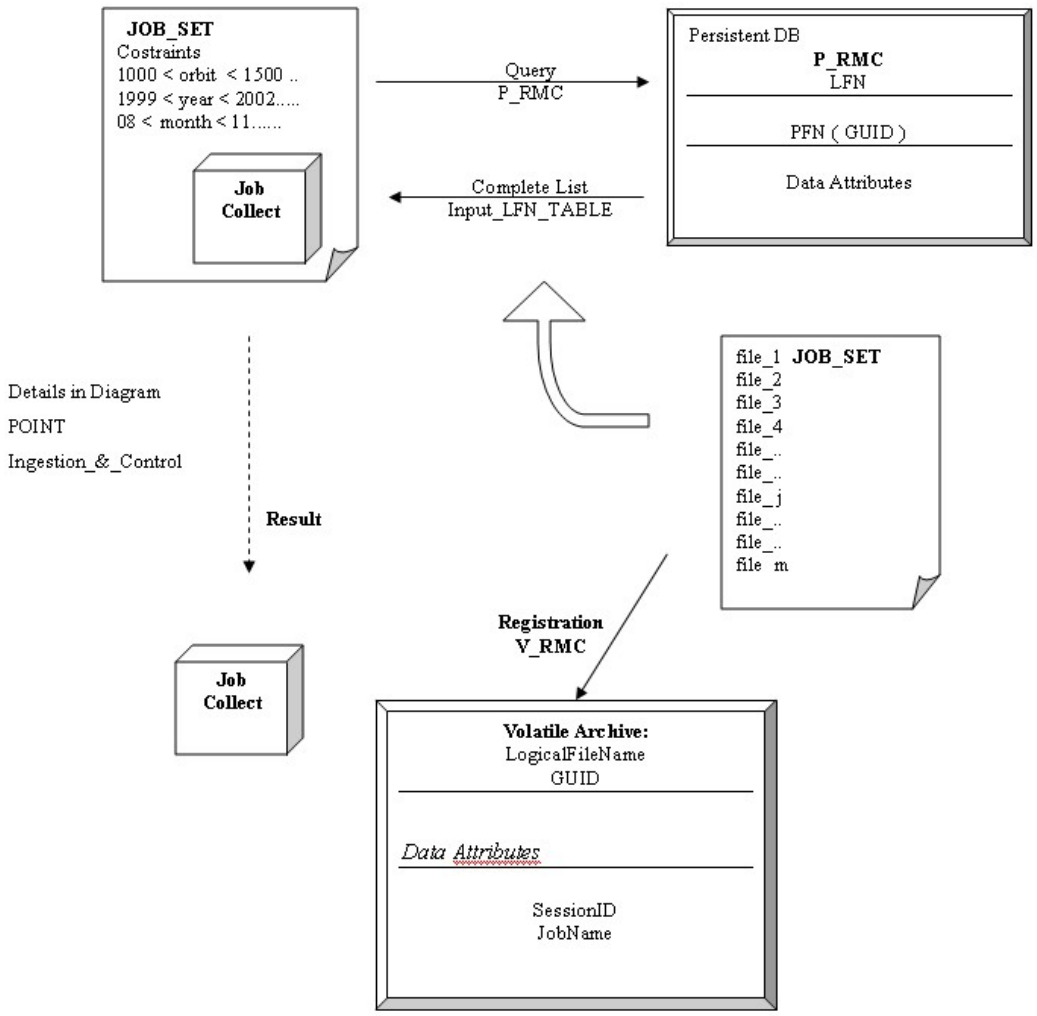

Figure 17: *The Collect GE-Job: during upload of data the Collect registers values on Cache Catalog.*

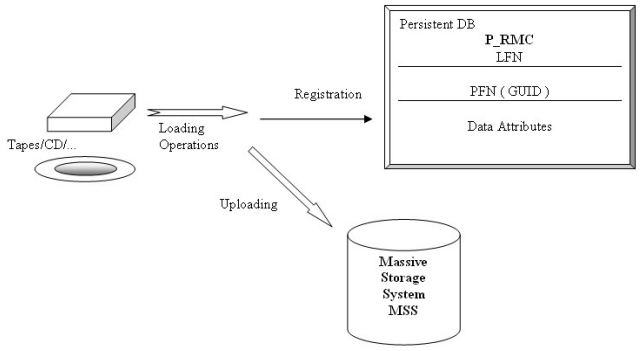

Figure 18: *The ingestion stage: from external storage system to the EGEO.*

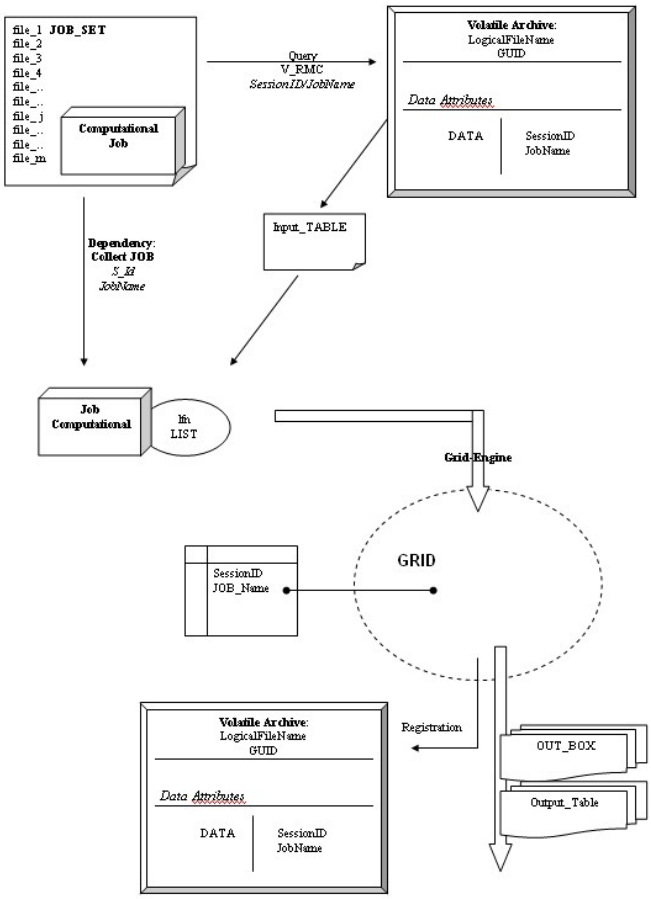

Figure 19: *The ingestion stage: the Volatile Archive is the main component for the EGEO cache.*

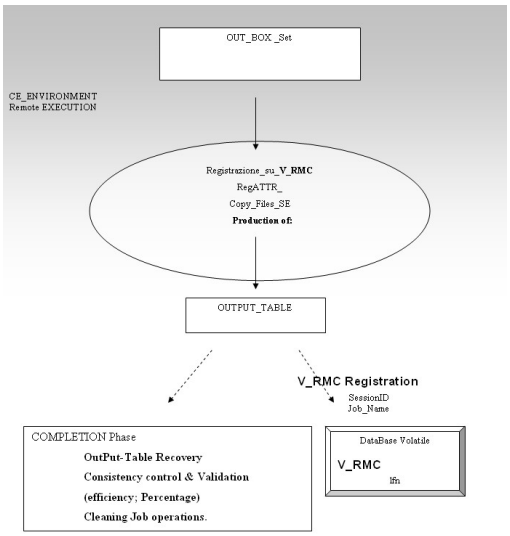

Figure 20: *The completion stage for GE-Jobs on the EGEO.*

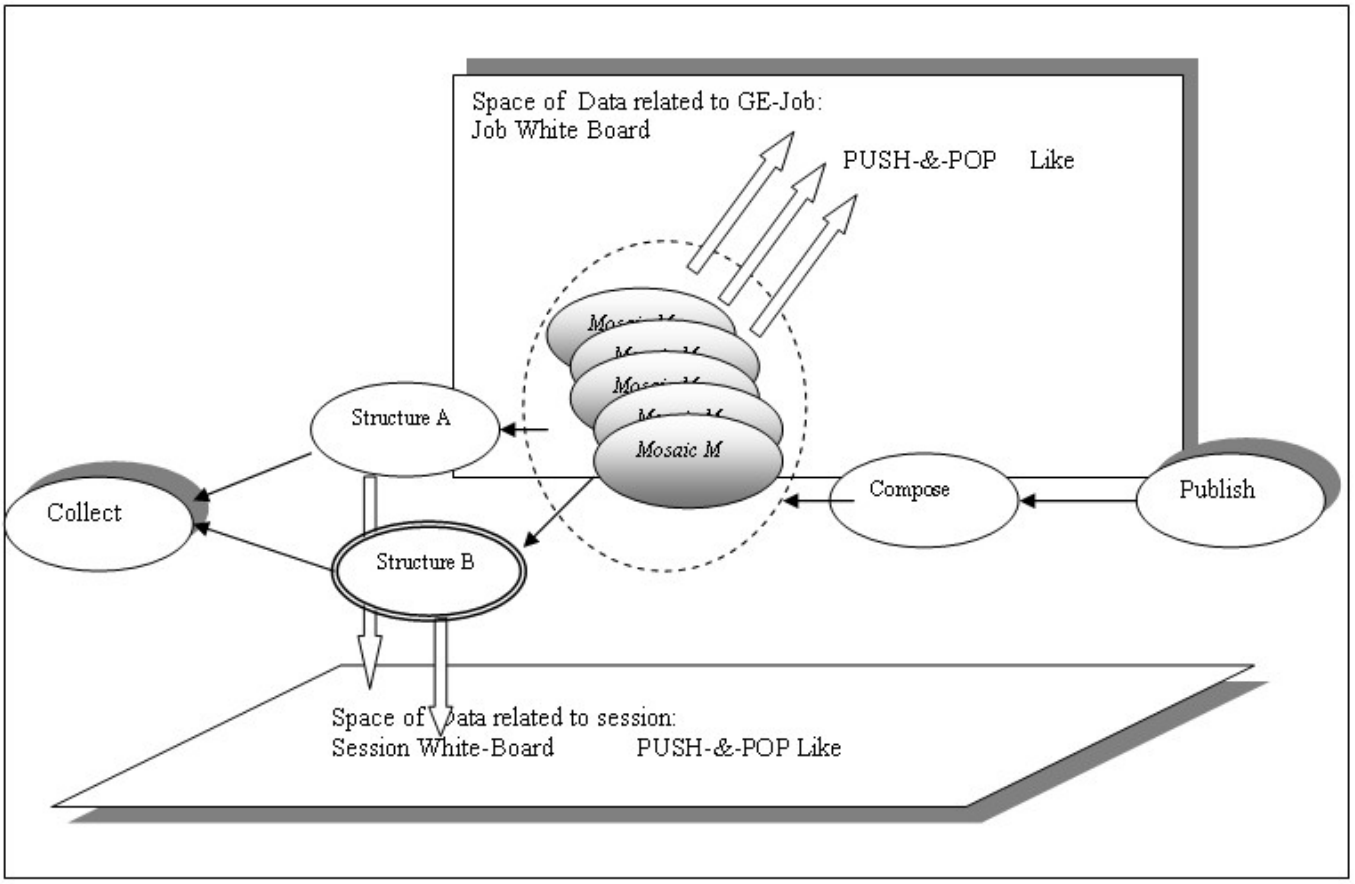

Figure 21: *How in the EGEO appears the Mosaic project. The schema illustrates the Mosaic but many tasks have a similar constitution.*

Table 2: The GE-Job Table: All available for the EGEO utilization through GridOnDemand gateWay.

BEAT	BESTcalibration	CNR	Collect
Convert	EnviProj	eoDataViewer3	eog-binning2
gome_meris	GOMEvalidation	GOMOS2L	HDFconvert
IceContour	idl_HRIT	Imager	JRC
JRCconcat	JRCregrid	L3aggregation3	L3binning3
L3Jap	L3sst	L3tcamean3	MedComp
MerisFR0	MGVI_JRC	ModisSpatial	ModisSST
mosaicCom3	N1Converter	ObjAn	old_jobs
Publish	RA2Raies	remapMeris3	RiverLake2
TCAWT	Vomir	WarpImage	WMSpublish
zBEST			

Figure 22: *The image captured during the browsing on Web Portal.*

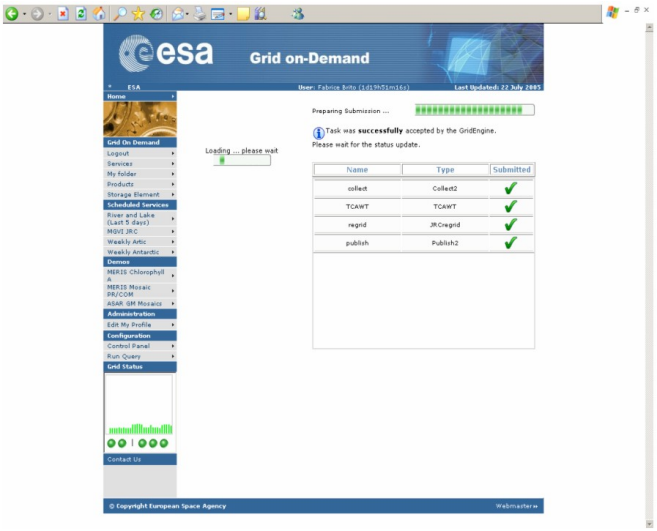

Figure 23: *The instantiation phase.*

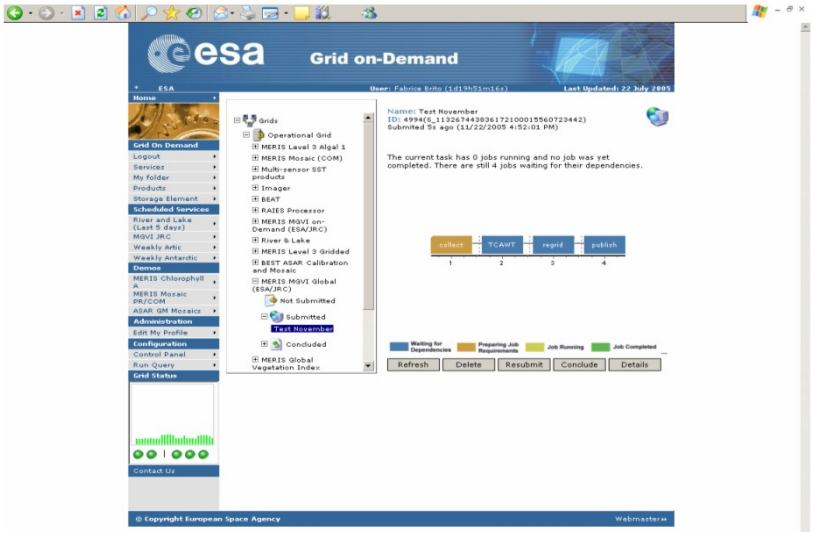

Figure 24: *How the GE-Jobs constituting the chain operate in a linked flow.*

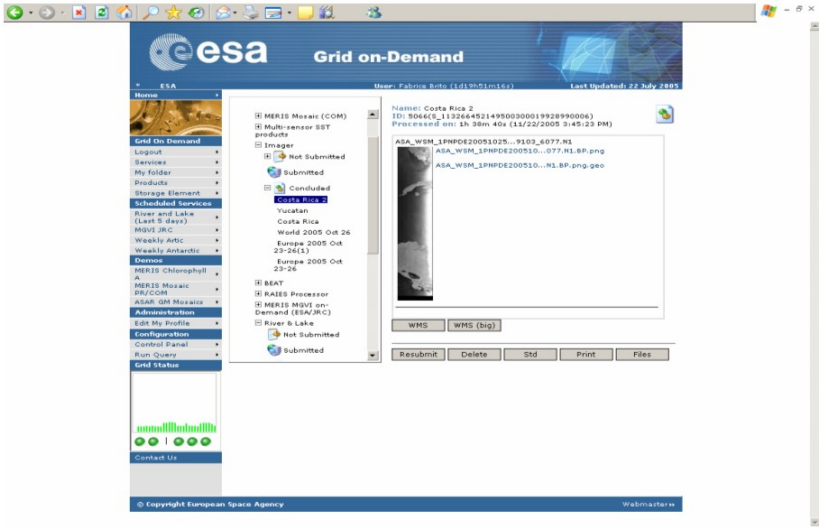

Figure 25: *How the in the EGEO the result are presented as Thumbnails and with the URL of their complete product.*