

Proyecto Fin de Carrera

Título	Automated tools and techniques for distributed Grid software
Subtítulo	Development of the testbed infrastructure
Autor	Carlos Aguado Sánchez
Tutor	D. Alberto Di Meglio
Ponente	D. David Fernández Cambronero
Tribunal	
Presidente	D. Ángel Fernández del Campo
Vocal	D. David Fernández Cambronero
Secretario	D. Francisco Javier Ruiz Piñar
Suplente	D. Luis Bellido Triana
Fecha	
Calificación	



Automated tools and techniques for distributed Grid software

Development of the testbed infrastructure

Carlos Aguado Sánchez



D. Alberto Di Meglio

IT Department - Grid Deployment

CERN *European Organization for Nuclear Research*



D. David Fernández Cambronero

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Madrid

Resumen

En los últimos tiempos, el Grid está cobrando cierta importancia como nuevo paradigma para la compartición y uso eficiente de recursos en todo el mundo. Su propósito a largo plazo es convertirse en el sistema automatizado de distribución de recursos computacionales que permita a organismos e instituciones aunar esfuerzos en la consecución de un objetivo común.

Sin embargo, la gran potencialidad de esta solución hace que su diseño requiera un alto grado de abstracción, definiendo una pila de protocolos distribuída que proporciona servicios y meta-servicios deslocalizados. Así pues, frente al planteamiento teórico inicial, han surgido distintas alternativas centradas en satisfacer distintos requisitos y cuya interconexión no es siempre posible.

Como parte de los esfuerzos en su desarrollo, diversas comunidades internacionales han percibido la necesidad de estándares que permitan la interoperabilidad entre estos sistemas de una forma nativa. Este es el caso del Open Grid Forum y sus diversas iniciativas (p.e. OGSA).

Asimismo, el desarrollo de cualquier distribución Grid se debe enfrentar a los clásicos problemas a los que se enfrenta el desarrollo de cualquier software distribuído (fiabilidad, disponibilidad, calidad de servicio, gestión, etc.). No obstante, en el caso del Grid, la masiva proliferación de distintos servicios como forma de provisión de características básicas hace que aparezcan múltiples relaciones entre ellos, lo que aumenta la complejidad del proceso de desarrollo.

Este proyecto fin de carrera se desarrolla en este entorno bajo la acción de los proyectos europeos ETICS y OMII-Europe. Trata de dar respuesta a la pregunta de *qué mecanismos establecer para que el desarrollo de software para Grid integre de forma automática en el mismo proceso de release las actividades conducentes a asegurar la interoperabilidad con otras distribuciones y por tanto ciertos niveles de calidad*. Y para ello, se ha realizado como prueba de concepto el test automático de la recomendación del OGF para envío de trabajos a entornos Grid (OGSA-BES) sobre la implementación de este servicio en la distribución Grid gLite (CREAM-BES).

La realización de dicho test se ha realizado sobre la plataforma ETICS, diseñada para gestionar y facilitar el proceso de desarrollo de cualquier producto software en general. Dicha plataforma incluye los elementos necesarios para almacenar toda la información correspondiente a componentes y sus relaciones, repositorios de código, plataformas soportadas, tests y elementos de validación que permiten implementar un proceso integral de calidad.

Los resultados muestran que aún es pronto para hablar de compatibilidad con la recomendación y por tanto asegurar la interoperabilidad. Sin embargo,

a pesar de los estados tempranos de los desarrollos, también muestran que la integración de la validación automática es viable. Ello ofrece una importante capacidad de análisis en el momento de comprobar el estado del desarrollo, lo que redundará en la mejora de la calidad final.

Keywords

Grid, SE, software engineering, Test, QA, quality assurance, compliance testing, functional testing, conformance, validation, job submission, testbed, automatic testing, virtualization, coscheduling, OGSA-BES, ETICS, OMI-Europe, gLite, CREAM, UNICORE.

Acknowledgements

To accomplish successfully the objectives stated in this master thesis I have needed the help of many people to cover many of the different aspects of its development.

Firstly, I would like to thank D. Alberto Di Meglio for his support during my stage at CERN and his big efforts to make me understand the current Grid environment, works, trends and international efforts. Due to him, I have travelled a lot and known a lot of exciting people around Europe, what will be interesting for the nearest future.

Secondly, I would like to thank for the support of the whole ETICS team based at CERN: Meb, Marian, Lorenzo, Tomek and Danica at the last part. We have shared many good moments and I have learned many things from all of them. I really enjoyed.

I am specially grateful to Guillermo and Rosa. With them, this year has been much more easy. They have taught me not only how to accomplish some technical tasks but the most important, how to face up with the changing environment we were working. Without question, it has had a big impact on these results.

I will be always grateful to all the people have aided me with specific but important issues at CERN and INFN: María, Maarten, Omer, Joachim and Paolo.

Another special mention must be made to my thesis tutor, D. David Fernández Cambroner. I thank him for his support during last years and his willingness to fill the paperwork during these years in the University.

Finally, but not the least I cannot stop thanking the affection and support of my parents, José Luis and Mercedes; my sister, Gloria and all my friends. Despite they do not know it, they are also responsible in some way of the completion of the work I present here.

Carlos Aguado Sánchez

Contents

Introduction	xix
1 The Grid	1
1.1 Computing grids	2
1.1.1 The power grid	4
1.1.2 The Grid era	5
1.2 Design principles	10
1.2.1 Core principles	10
1.2.2 Architecture	13
1.3 Results: Grid potential	17
1.3.1 Distributed supercomputing applications	17
1.3.2 Real-time distributed instrumentation	19
1.3.3 Data intensive computing	19
1.4 Implementations	20
1.4.1 The Globus Toolkit	20
1.4.2 gLite Middleware	24
1.4.3 UNICORE	29
1.4.4 OGSA initiative	31
1.5 Summary	34
2 State of the Art	35
2.1 The software quality process	36
2.1.1 Quality metrics	37
2.1.2 Software testing	38
2.2 Description of a testbed	48
2.2.1 Distributed testing	49
2.3 Existing solutions	50
2.3.1 The Apache Software Foundation	51
2.3.2 Sourceforge	52
2.3.3 Metronome	53
2.3.4 Others	55

2.4	Summary	57
3	The ETICS system	59
3.1	What is ETICS?	59
3.1.1	Objectives	60
3.1.2	Features	61
3.2	ETICS architecture	62
3.2.1	Data model	63
3.2.2	Service architecture	71
3.3	The ETICS facility	78
3.4	Summary	80
4	Test use case: OGSA-BES compliance	83
4.1	OGSA-BES description	84
4.1.1	Data model	85
4.1.2	Port-types	88
4.2	Test objectives	92
4.3	Summary	94
5	Test use case: design, implementation and results	95
5.1	Test design	96
5.1.1	The client	96
5.1.2	The testbed	99
5.2	Test implementation	102
5.2.1	Compliance test code	102
5.2.2	Deployment of the scenario	103
5.2.3	Modelization in ETICS	106
5.3	Test deployment and results	109
5.4	Summary	113
6	Conclusions	115
A	Test implementation	121
A.1	ETICS commands	121
A.2	Testsuite design	122
B	Screenshots	127
B.1	Modelization in ETICS	127
B.2	Results	127
	Bibliography	138

List of Tables

3.1	Module parameters	65
3.2	Project parameters	65
3.3	Subsystem parameters	65
3.4	Component parameters	67
3.5	Configuration parameters	67
3.6	Inherited module parameters	68
3.7	VCS Command parameters	70
3.8	Build Command targets	70
3.9	Test Command parameters	72
3.10	Roles	72
4.1	BES Factory attributes	89
4.2	BES Activity attributes	91
5.1	Test methods results	111
A.1	Test methods description	123

List of Figures

1.1	Globus toolkit	22
1.2	gLite architecture	26
1.3	UNICORE architecture	30
2.1	V Model	39
3.1	ETICS Architecture	74
3.2	The ETICS facility	78
4.1	BES state model	86
5.1	Compliance test results	110
B.1	Compliance test tree	128
B.2	ETICS job submission	129
B.3	Compliance tests execution	130
B.4	Repository deployment	131
B.5	Worker node deployment	132
B.6	CE deployment	133

List of Acronyms

AFS	Andrew File System
AUTHN	Authentication
AUTHZ	Authorization
CA	Certification Authority
CE	Computing Element
CERN	European Organization for Nuclear Research
CREAM	Computing Resource Execution And Management
CRM	Customer Relationship Management
CSS	Cascading Style Sheet
CVS	Concurrent Versions System
EAI	Enterprise Application Integration
EGEE	Enabling Grids for E-ScienceE
EPR	Endpoint Reference
ERA	European Research Area
ERP	Enterprise Resource Planning
ETICS	e-Infrastructure for Testing, Integration and Configuration of Software
EUGRIDPMA .	European Grid Policy Management Authority
GGF	Global Grid Forum

GSI.....	Globus Security Infrastructure
GUID.....	Grid User Identifier
HEP.....	High energy Physics
ICT.....	Information and Communication Technologies
IEEE.....	Institute of Electrical and Electronical Engineers
INFN.....	Istituto Nazionale di Fisica Nucleare
INI.....	Initialization file
ISO.....	International Standards Organization
IUT.....	Implementation Under Test
JCR.....	Java Content Repository
JSDL.....	Job Submission Description Language
LFN.....	Logical File Name
LHC.....	Large Hadron Collider
LRMS.....	Local Resource Management System
LSF.....	Load Sharing Facility
MCDC.....	MultiConditional Decision Coverage
MTBF.....	Mean Time Between Failure
MTTR.....	Mean Time To Repair
NSF.....	National Science Foundation
OASIS.....	Organization for the Advancement of Structured Information Standards
OGF.....	Open Grid Forum
OGSA.....	Open Grid Services Architecture
OGSA-BES..	OGSA Basic Execution Service
OGSA-DAI...	OGSA Data Access and Integration

OMII	Open Middleware Infrastructure Institute
OOP	Object Oriented Programming
PBS	Portable Batch System
PKCS.....	Public Key Cryptographic System
PKI.....	Public Key Infrastructure
POSIX	Portable Operating System Interface
QA.....	Quality Assurance
RBAC	Role-Based Access Control
RDBMS	Relational Database Management System
RSS.....	Rich Site Summary/RDF Site Summary/Really Simple Syndication
SAML	Security Assertion Markup Language
SCM.....	Supply Chain Management
SE	Storage Element
SLA.....	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSO.....	Single Sign On
SVN	Subversion
TSL.....	Task Sequencing Language
UDDI.....	Universal Description, Discovery and Integration
UNICORE ...	Uniform Interface to Computer Resources
UoW	University of Wisconsin-Madison
VCS	Version Control System
VO.....	Virtual Organization

VOMS.....	Virtual Organization Membership Service
WA.....	Web Application
WN.....	Worker Node
WS.....	Web Service
WSDL.....	Web Service Description Language
WS-GRAM...	Web Service Grid Resource Allocation and Management
WS-RF.....	Web Service Resource Framework
WSRP.....	Web Services for Remote Portlets
XACML.....	eXtensible Access Control Markup Language
XML.....	eXtensible Markup Language
XSL.....	Extensible Stylesheet Language
ZSI.....	Zolera SOAP Infrastructure

Introduction

Grid computing is becoming more and more important in the current days as a possible solution for the increasing needs of resources in terms of power computing and storage capacity for the current research challenges in the world.

This new paradigm offers very amazing views for the ways in which the new generation information networks will be established and managed as well as the new highest performances ever seen. However, its difficulties are also higher than before because this new perspective needs very abstract and complex tasks to perform high-end, general and extensible results. This new environment is mainly based on the wide distribution of services. Services that provide high-end features to a global infrastructure and delivered through different partners and different sets of non-homogeneous resources.

The development of this kind of software is a very complex task. The different components of the services will run under different platforms, environments and, the most important, will be implementations from different teams or companies so here appears the main difficulty to this challenge: how to ensure the perfect understanding among all the peers? or in other words, how to integrate all the different software developments for achieving a acceptable global quality?

The development process of this kind of software consists on the following phases:

- development: producing pieces of software
- test: checking the software against different quality patterns
- integration: joining to other components
- release: producing public artifacts
- configuration: customizing the component for particular usage

In the case of Grid software, the current approach consists on satisfying all the previous requirements by defining sets of compliance rules, so the different

software parts should understand by itself the own definition. However, the great amount of different combinations of hardware/software (so called platforms) makes impossible to manage the whole release process in an extensible, reproducible and high-quality way unless automatic tools exist. So it is clear that new tools are needed for automating all the process of release a distributed software in the Grid.

These new tools include not only procedures for building automatically the software components but also mechanisms for making this process based on software engineering techniques that guarantee the proper continuous integration and testing based on different kind of schemas.

The main objective of this work is to provide a clear idea about the interoperability issues to be concerned when developing distributed software for Grid infrastructures and show how some existing works can help providing automated tools for achieving reasonable quality results at the end of the process. The main question stands as how can be guaranteed that several implementations of the same concept to be interoperable, apart from the specific details of each one.

This is not a trivial issue since until now, there are plenty of Grid distributions, not always ready to perform common tasks among them. Right now, the development process for these scenarios is quite complex and requires efforts that could need to be repeated to extend those characteristics to others. Hence, from this point of view, the definition of a common input interface (API) is something valuable for providing future interoperability. However this API exists (just now being under construction OGSA-BES), its definition by itself does not guarantee the desired goal and will be necessary to implement ways to check the compliance on real scenarios, checking so the real interoperability. If the interoperability can be achieved by this mean, anyone could write a simple and generic client to connect to any computing service in order to send jobs and getting artifacts under different platforms.

The study will start analyzing the origin of all the trouble: the development of software for Grid infrastructures. In this section, the use cases of the Grid solution like the new concept of parallel services based on the co-scheduling of tasks and the new paradigm of sharing resources in a efficient and controlled way will be deeply explained. The consequences in the distributed software development will raise: the use of methodologies, metrics for quality assurance and its changes due to the new paradigm. Also, some existing tools with their pros and cons will be introduced.

As part of the description of the environment, the state of the art in software engineering techniques and tools for distributed Grid development will be presented (2). Within it, the common approaches, current trends and open source tools will show why is quality assurance important to achieve

successful results and the mechanisms oriented to that.

Once understood the underlying trouble and the current challenges, the ETICS project is depicted as a way to address these new issues in an automatic way (chapter 3). This project aims to provide a build and test service over a pool of several platforms, generating reports and publishing the results in a central repository for later integration in other projects.

The work exposed in this thesis covers automatic deployment of virtual scenarios to verify interoperability between services based on standard compliance testing. Therefore, chapter 4 describes the recommendation used in the test, drawing the main objectives of the test.

Finally, after the description of the global environment and trends, the current initiatives and once focused on the field of study, the chapter 5 explains procedure and actions accomplished to achieve the previous results. It covers the design and implementation phases of not only the test but also the testbed where it has place. It also shows how the test (i.e. its development and deployment) is modeled within the ETICS facility. Chapter 6 shows the conclusions of the tests, as well as found issues during the process. It is also shown the natural path of activities to be performed after this work.

a ella

Chapter 1

The Grid

The term 'the Grid' was coined in the early 1990s as an advance distributed computing infrastructure, sharing multiple computing resources along Internet. But despite previous initiatives for defining new ways of distributed computing, it was in these moments when Ian Foster, Carl Kesselman and Steven Tuecke brought together the concept of Grid in the modern term. The main idea was to establish the proper infrastructure that allowed access to computing resources as easy as the access to the energy in the electricity distribution. And not only purely computing resources but also storage and network resources within the proper security framework.

This new kind of technologies has allowed not only that new range of scientific and engineering problems could be addressed (like data intensive computing, distributed supercomputing or high throughput computing) but also new business models could be exploited. However, this new perspective represents a complex challenge: to define the set of technologies which allows to abstract the underlying specific ones (platforms, networks, etc.) in a completely transparent and scalable way for developing Grid-wide applications. As in the power grid, every device which needs energy connection must be compliant with a set of rules specified in the contract, the use of computing grids establishes some constraints in its design in order to get the same scale of flexibility and scalability as in the electrical model. With this new approach of computing resources usage, new troubles appear concerning the design respect the classical software development model: very high level of abstraction, need of clarifying the services provided by each entity and development of the proper tests (distributed along all the framework).

So along this chapter, grid computing will be presented covering the following aspects: the starting point and its analogy with power grids, the initial proposal (the core design principles and underlying architecture to achieve them), consequent results and its potential and finally the current

initiatives and implementations of such theoretical model.

1.1 Computing grids

During the History, it is proven the big development of societal structures is bound up with the deployment of supporting infrastructures. Most of the trade thriving cities in the world emerged out of the rise of their infrastructure. Usually, the improvement of these ones allows to get profit of the natural resources and opportunities, getting a competitive advantage. That advantage enables to point new challenges and risks that others cannot afford and therefore make progress into its knowledge field.

In case of information technologies and computer science, this has happened in the same way. The use of first computers let face automation of information, in a faster way than before. This new information management allowed to design new techniques for taking advantage of faster calculations made by computers and to deploy a completely new way of thinking, abstracting implementation issues (relying on the infrastructure) and being centred in the specific topic.

The networks appearance involved the possibility of making that information world-wide available, therefore enabling new ways of information management, communication and business. This improvement made possible teams to joint and work as communities, pointing to the establishment of new collaboration relationships, what, again, allowed to deal with more and more complex problems. Now, Internet works as a big social network which is useful not only for science and engineering tasks, as in these first moments, but also as the place where getting in contact, making businesses, distributing media content or simply as entertainment source.

The current technological development has favoured very successful results in many knowledge fields (cultural, economical, social, etc.) until now, but the crucial fact is that these technology improvements are so deeply-rooted into the society that new challenges have been discovered and can be theoretically addressed. For example, market modeling and simulation, marketing data mining (or data warehousing), weather analysis and simulation, protein synthesis modeling (proteomics), genomics (genoma analysis and sequentiation), etc. are new cases in which theoretical models have been developed thanks to all the new range of information technologies. However, these new skills need very strong capabilities, not only computational resources but also storage or high quality network resources, which are not always available. Therefore, something that was a tool to manage the complexity of some tasks has been obsoleted in a very short period of time.

The new techniques discovered with these tools demand now more powerful computing elements to achieve the desired goals.

Therefore, next step consists on searching new perspectives in science and engineering that allow to increase the efficiency in use of current resources, enlarging the global systems capacity. During this past experience, two phenomenon showed that this new perspective should be centred into collaborative efforts: the establishment of the computer science market and the proliferation of scientific communities with common objectives that Internet puts together.

Computer market evolution during the last decades is characterized by the remarkable growth of computing and storage capabilities for the same prices (or even less). In this way, what was common computing equipment during the 1980s in the main scientific and public research laboratories computer centres now is comparable to computing equipment for domestic use. This important fact has made that now it is quite common to find different kind of service providers, offering services based on this commodity hardware and at very competitive prices. This means that the potential capability of tools that allow to use aggregation of resources grows exponential with the available number of users¹.

Also, another important fact is that great science and technical developments have been always linked to public initiatives for research and innovation in the different knowledge fields. Nevertheless, many times these research jobs were accomplished in isolated teams, with no mutual or so often connections. Internet appearance with its new revolutionary characteristics (ubiquity, speed and versatility) made possible the idea of joining research teams, improving multidisciplinary collaborations and global growth.

Under this context, in the early 1990s, Ian Foster, Carl Kesselman and Steven Tuecke proposed a new architectural model based on distribution of services. Distributing computing was proposed some time before and was already being used (e.g. Distributed Computing System [1]). Their new idea was developed on the basis of some new infrastructure that enables full and dynamic management of resources across and multiple organizations. This concept was based on the same one used since many years before in electrical power suppliers: the power grid.

Power Grid concept consists on defining the proper system architecture which is designed to provide computational resources in the moment they are needed for problem solving. But this concept is slightly different from classical distributed computing. It relies on the same principles than have

¹Conclusion stated in the Metcalfe's Law: *the value of a telecommunications network is proportional to the square of the number of users of the system.*

been successfully implemented in the power grid, predisposing to the success of this new approach.

1.1.1 The power grid

The widely use of electrical energy and consequently the electricity distribution network expansion (formerly known as power grid) is one of the most important elements that have determined the human development during the 20th century. Mainly, its model has been the responsible of this success, allowing this energetic source to have one of the biggest penetration rates among all potential users in the world. The success of its model lies in two elements:

- Clear producer-consumer abstraction and separation from resources
- Standardization in network access and service supply

Electrical energy is a kind of energy that can be obtained from chemical, mechanical, thermal or another electromagnetic sources. The choice of one or another source depends on economic, ecologic, raw material availability, etc. reasons. However, whatever source is chosen, it provides the same final result: electric energy. In addition, the great versatility, easy use and economy of scale characteristics have made this model one of the most common ones for power supply in the world. It is based on three main actors: producers, consumers and suppliers. These last ones, play the key role in the model because they provide the resources from different kind of producers to a very big community of users across a very wide area under the desired quantity and quality.

One of the troubles suppliers have to solve is to put in contact high heterogeneous producer systems (managed by different companies) with a huge consumer market in which quantity, quality and affordable price are very different. In this environment, the second distinguishing characteristic of power grids appears: virtualization of interfaces (standardization of technologies), generalisation in the access and pricing policies. Again, this one has appeared as a very important point in the model because if some customer needs the power grid services, it is only needed apply for service provision under the specified contract. After, the client will be able to connect to the power line any compliant device with the arranged interface and will be charged depending on his consumption. So power supplier manages automatically production and consumption issues in terms of quality, quantity and billing (accounting).

Therefore, resource virtualization (producers abstraction) and information virtualization (service supply standardization) are the main principles to reach in order to define the computational systems architecture which allows the flexibility, versatility and potential offered through power grids.

1.1.2 The Grid era

Like at the beginning of 20th century, when the electric power grid and the associated technologies represented a revolution², now Grid computing is a clear evidence of another revolution as part of the Information Society. The dramatic increase of computer and network capabilities and the years of research in metacomputing have created the required basis for the next step in the evolution of information technologies.

The Computational Grid is defined as the hardware and software infrastructure that enables the management of large-scale pool of resources, providing dependable, consistent, pervasive and inexpensive access to high-end computational facilities:

dependable because for using applications above the Grid it is fundamental that performance issues will be solved in terms of predictability and sustainability. Grid must provide high-end levels of quality of service including network bandwidth, latency, jitter, computer power, security and reliability.

consistent because for service provisioning it is a fundamental concern to provide standard services, being accessed through standard interfaces and with standard parameters. The challenge is to abstract the underlying heterogeneity of systems without compromising high-performance execution.

pervasive because it is necessary to count on services always available within the specified environment. This requirement supposes that grid resources will be available under the environmental conditions have been settled.

inexpensive to facilitate wide access and broad distribution at competitive prices.

Under this conditions, Grid can find its opportunity for success because it will be able to:

²The biggest revolution after the industrial one. It provided reliable, low-cost access to a standardized service, making electrical resources universally accessible.

- seize the technological improvements in terms of computational capabilities
- increase demand-driven access to computational power, using high-end virtual dedicated resources for specific and demanding purposes
- increase utilization of idle capacity
- improve sharing of computational results, making results available to big interested communities in an easy way (e.g. weather forecasting)
- enable new problem-solving techniques and tools like network-enabled solvers or teleimmersion techniques

But despite these strengths, the real powerfulness of Grid lies on its synergistic ability for using high performance computing, networking and advanced software in order to provide universal access to advanced computational facilities, in the secure way that allows not to consider the location of users and resources³.

Grids usage

The use of distributed systems for metacomputing purposes has shown so far that applications development lacks of important characteristics as costs, reliability, security or scalability. Grid will achieve its potential and usefulness if the proper models and techniques are developed within its deployment. It is necessary to design the programming models (APIs), tools and standards for applications which allow to put the same abstraction layer used in standalone computing⁴ into practice.

This perspective establishes the following role-based hierarchy for the deployment and usage of Grid technologies with software engineering quality:

Grid developer is responsible for implementing the basic services which provide simplicity, connectivity and security using local system services.

Tool developer is concerned about developing tools, compilers and libraries in order to implement the Grid services where application developers will develop the grid-enabled applications. He is responsible for providing efficient implementations of after-used programming languages.

³What has an important analogy with the globalization concept.

⁴Abstraction from assembly language to high level languages or from one-off libraries to application toolkits.

Application developer will write grid-enabled applications using the underlying grid technology. He will use supported programming models (languages, libraries and tools) and services (for security, resource discovery and brokering, data access, etc.) to design complex algorithms and high-end applications.

End user will make use of all these grid-enabled applications for making use of grid resources and services in chemistry, engineering, biomedical sciences, etc. They will require the applications to be reliable, predictable, confidential and usable.

System administrator must manage the infrastructure on which grids operate. This is a complicated task because in order to get grid usefulness, locally managed resources have to be available in the global community. So this political issue must be solved delocalizing administration and automating multisite issues.

Communities using Grid

Usefulness of the Grid makes it adequate for different scenarios. Not all of them have the same objectives and requirements (in term of resources), but the following examples are clear target environments under which Grid technology may supply great improvements in case of adoption.

For computational scientists and engineers, Grid technologies may suppose the proper tool to simulate and visualize in real time their applications (steering and controlling the resources). Without it, any error only will be detected after the collection of results, which is not acceptable for long time-consuming simulations.

Experimental and environmental sciences need new control resources technologies like Grid to manage and use remotely large devices which are not under their physical control. Use of big instruments like telescopes, distributed weather stations or particle accelerators is not possible unless it is done under automatic systems that control the whole process and enable concurrent access to results like Grid. Also, the other side of these collaborative studies is the management of results produced with these instruments and the knowledge base generated. The multidisciplinary, dynamic, nonlinear and multitime-scale nature of some problems implies that shared-enabled tools like Grid are the only way to gather all the experts for studying the problem.

Nature associations (scientific, business, research, etc.) are another target group that could be interested on Grid technologies because sometimes not all the members have the same capability and accessibility to resources. So

Grid appears as the manner to enable controlled access based on the policies settled inside the association.

All global companies, specially ICT related ones, could make the most of Grid, using it for improving their corporate tools like intranets or even their Enterprise Application Integration (ERP, CRM, SCM, etc.). Use of such technology could make easier the whole integration in the value chain improving the global management and consequently, the global results.

One of the more amazing applications could be development of teleimmersive applications which would enable full remote training and education. In this case, the implications in the current educational models would be huge because of the possibility of giving access to the previously unaffordable resources (in terms of time and location).

Governments are another important target for the Grid implementation. Many times, they are responsible for disaster response, national defense or long-term research. Using grids, national resources occupation would improve the strategic response reserves and nationwide collaborations, given any threat or concern.

All these use cases share the same technical objective: use high-end computational resources for problem solving. However, the differences in the overall results and the Grid availability to afford them make the Grid to be the most versatile and powerful tool that can contribute to the growth of every field.

Architectural entities

To support the Grid concept, definition of the proper architecture is something fundamental. The new architecture of services must provide answer to the following questions: how to support high platforms heterogeneity? how to cover all different specification requirements? if the current computer science perspective points to fractal definition of services, which are the implications of this approach for the service provision and the relationships between all provider elements?

For addressing these questions, the architecture model can be split in three different semantic abstractions: functional, structural and process-oriented abstraction [2].

Functional abstraction comprises the way in which system relates with the environment providing useful results. The way to achieve this goal is extending the basic services provided by the conventional computing during last decades to Grid computing environments. The main services to implement are: process management, authentication and authorization, communication, naming and addressing, data storage, security and accounting.

Structural abstraction refers to subsystems and relationships between the elements inside the Grid. Within this model, Grid articulates four different entities which are supposed to provide different kind of services, not overlapped and in increasing order of abstraction. From the lowest level in this stack, the end-systems level (providing platform access based on OS functionality), cluster level (set of resources providing integrated resources access with enhanced capabilities), intranet level (entity which summarizes the whole pool of resources of a specific organization and provides unified access to high-end organizational resources) and Internet level (provide internetworking computing resources spanning across multiple intranets and solving localization, control and internationalization issues).

Finally, process-oriented abstraction refers to the inner dynamic activity inside whole Grid infrastructure. For that, a set of interfaces has been defined. Services and protocols that allow to implement the global computer-based infrastructure seamlessly. The main set of services that need to be implemented consists of control and signaling processes, scheduling for allocation and use of resources (process-based access), shared space addressing, resource allocation policies and brokering mechanisms (enforcing policy decisions points).

Challenges

As shown before, the main objective of Grid computing is quite ambitious: enable global access to resources satisfying owner or local requirements. However, the start-up of this infrastructure depends strongly on several levels of computing resources and techniques that are currently under research.

Several topics involve new challenges in computer sciences. Others aim to change the current perspective of technology, evolving to new ways in which computers are used.

1. Information Society contribution: nature of applications, algorithms and problem-solving methods or instrumentation and performance analysis.
2. Technical challenges: system architecture, network protocols and infrastructure, resource management or end systems.
3. Knowledge challenges: programming models and tools or security models.

These ones are some examples of what current Grid requirements will involve in the development and deployment of computer science in particular

and Information Society in general. In the next sections, the principles and core elements that will enable the evolution towards these objectives are explained in more detail.

1.2 Design principles

In the previous section, the main idea behind the Grid is outlined. This presentation has made emphasis mainly in the social and scientific origins, showing the motivation for such tendency in the current research and the desirable results. In this section, Grid is presented in a more technical way, its principles and core features. These basic principles consist a small set of concepts that being extended properly, provide analogous functionality, in abstraction terms, to design features in power grids: virtualization and provisioning.

1.2.1 Core principles

Term virtualization refers to the abstract capability of underlying infrastructure to separate functionality generation from resource management and so, support a broad range of resources, providing the same usefulness. In Grid environments, this feature provides the essential capability for pooling together individual resources into the big platform available to consumers. The proper design of the abstract stack is the main goal in Grid development in order to render the particular needs without showing how the accomplishment takes place. Therefore the virtualization techniques will supply the general low-level user requirements, taking advantage of the resources heterogeneousness but avoiding constraints imposed by specific implementations. This concept explains what Grid environments provide in terms of infrastructure and capabilities. However it is also needed to know how this resources can be used, what introduces the second key concept, the provision of services.

Provisioning means the set of services and available mechanisms which define the behaviour of resources in terms of policies of access and use, resource allocation and life cycle management (resource monitoring and control).

Both, virtualization and provisioning are covered under the global concept of Virtual Organization. This one is the entity which constitutes the entry point to resources and is responsible for granting rights to consumers upon fulfillment of corresponding policies. In one hand Virtual Organizations summarize available resources showing them to final customer as set of services. So the same physical infrastructure can perform several operations

transparently to the user, providing different quality of service patterns simultaneously. In the other hand, they provide secure mechanisms for enabling dynamic access to low-level resources enforcing the predefined policies which control the use of resources.

Until now, resources have been introduced as a general computer supporting unit, but in Grid environments, these ones are classified according to their abstraction level and their functionality in the stack of features. Therefore, in Grid there are three type of resources:

information constitutes the main assets which will be exploited using grid tools. Information comprises all the models and metadata which make the data meaningful. The way in which this information is stored or its source is not as important as to know the semantic relationships among them that show the state of that information and allow to exploit it. Within information models, another key point is the appearance of machine-readable standards which make possible introduction of applications to be used not only directly by humans but also by computers.

applications are the specific resources which provide the intelligence or business logic to perform operations over the data. Applications are provided as services or set of ones reflecting any level of complexity and invoking different relationships. One important aspect of deploying Grid-enabled applications⁵ is the reusability and flexibility, what appears as a result of the proper information models and relationships among entities and that allows the applications to be orchestrated in higher-end services.

infrastructures are the physical entity which delivers the computing, storage and networking capabilities. Infrastructures and related services supply the low-level features needed for upper layers inside the services stack. They comprise computing elements, communication elements, storage elements but also software components and protocols in order to get efficiency, reliability, low latency or encapsulation. Despite the high abstraction level provided by Grid technologies, the network infrastructure is still the key point to successfully deploy it, due to the required high quality of service.

But under the perspective of virtual organizations providing the features mentioned, interoperability and portability appear as the main issues to be addressed. In networking terms, interoperability involves to define the

⁵Term commonly used to refer to applications aware of the underlying scenario and hence, Grid capabilities

set of protocols and portability involves definition of standard application programming interfaces (APIs) which specify the standard behavior and speak the defined protocols. This model matches exactly with what in last decades has been called Service-Oriented Architectures.

SOA could be defined as *an application architecture within which all functions are defined as independent services with well-defined invocable interfaces which can be called in defined sequences to form business processes*. Its main goal is to achieve the so-called loosely coupled property. Loose coupling is the attribute of systems which, assuming modular design, reduces the dependencies across system components increasing flexibility by focusing on design of interfaces⁶.

Usually, the SOA paradigm can be explained as the composition of four entities which interact in the whole architecture:

services are well-defined and self-contained functions that do not depend on the state of other services and which also provide the semantic and the state management to capabilities offered. They constitute the gateway to applications and expose message-based interfaces suitable to be accessible across different networks.

messages represent the final definition of service requirements and results. They establish explicitly the contract between the producer and the consumer abstracting completely inner implementation details (improving the transparency).

policies are the set of instructions which govern the service provision. They are used to negotiate the quality of service, security and management issues related with the applications and utilisation of underlying resources.

states are the piece of information that enable requests chaining. Services are able to manage states ensuring their consistency and accurateness even in a durable way. The proper definition of state management can improve reliability of services and scalability of the global architecture.

The currently most extended approach to the SOA paradigm is based on the so-called *Web Services*. Web services are technically *a software system designed to support interoperable machine-to-machine interaction over a network*. There is a collection of technologies which implement the SOA model stated before. Most of them are implemented using XML-based technologies. The current core specification is compounded of the main components:

⁶The interface concept aims to reduce the risk that changes within one component create unanticipated changes within other components.

SOAP (*Simple Object Access Protocol*) is the protocol used for exchanging messages between services. It allows the use of several underlying implementations (called bindings) as for example, HTTP, HTTPS, SMTP, etc.

WSDL (*Web Services Definition Language*) is the protocol used to describe how to perform the communication with the web services in terms of input arguments and results obtained but in the abstract way that allows reusability.

UDDI (*Universal Description Discovery and Integration*) is the protocol to access a registry for Web services which allows to manage the information about the services provided by Service Providers, the service implementation (WSDL specification file) and related metadata.

This Web services framework presents two advantages for Grid development based on OSA which are:

1. fully support of dynamic discovery and service composition in very heterogeneous environments (introducing interface definitions, endpoint implementations and the mechanisms for registering and discovering them)
2. widespread adoption in current business and research environments what promotes existing services and tools can be exploited in order to facilitate the starting point for Grid applications

Once seen the concepts became important in Grid development, next section presents the real architecture that provides these features. The next description is not a detailed description of specific software implementations, but the general structure of resources (the protocol stack) which is intended to enable previous concepts.

1.2.2 Architecture

Grid architecture definition starts from the Virtual Organizations point of view because is the main entity for providing the desired virtualization and provisioning characteristics. From this view, the main task is automation and management of relationships from any potential user. So interoperability appears as the main goal to address and therefore the protocol stack.

Interoperability is the key point in order to guarantee the proper robustness and flexibility which allow relationships to be established among arbitrary entities. Otherwise, users or applications (on behalf of users) are

forced to enter into bilateral arrangements, what does not ensure extensibility to third parties. The way of achieving interoperability is across definition of standard protocols. These ones specify how distributed system elements interact with others and the information exchanged during this interaction, but focusing on external requirements and giving internal implementation details up. This also preserves control over local resources so individual organizations control still their own resources and user policies.

Another important point to design the proper stack of protocols is to consider its role in service provision. Services are defined by the protocol they speak and the functionality they implement. The way to design applications that use services is with APIs and SDKs. So if the stack of protocols is not properly designed, there are only two mechanisms to achieve interoperability: using a single implementation everywhere or knowing the details of every implementation. Both of them prevent the extensibility, portability or versatility provided by Grid applications.

Consequently, the open architecture for Grid computing to address those issues has been defined in [3] [4]. It is based on a conical model whose vertex is composed of the core protocols (a small number itself), different underlying technologies are under it and the high-end applications are mapped onto it. It is important to denote the analogy with current Internet model, where Internet Protocol is in the core, over it a broad range of different protocols for different purposes and under it the set of protocols that enable access to different physical resources. As this model has brought Internet expansion successfully, the same Grid approach should get the same success.

Fabric layer

The fabric layer is the lowest one in the Grid protocols stack so this one is the main responsible to provide access to the capabilities offered by resources. In this case, resources comprise physical ones (computational, storage, network, etc.) or logical sets of these ones (clusters, distributed file systems, etc.).

Of the two main characteristics of Grid, this layer mainly provides the virtualization property for higher level protocols so it must deal with the details of implementations of every resource. Thus, there is a subtle dependency between sharing operations that can happen at higher levels and the functions implemented at fabric one. Features provided by resources (i.e. the fabric layer) should be prepared for enquiry and management mechanisms. The first ones allow to browse the resources structure and eventually know their status and the second ones use of their capabilities under some controlled way to provide quality of service over demand.

Finally, some capabilities that should be provided at this layer are:

1. computational resources: monitor and control of processes, allocation, reservation, status, characteristics browsing, etc.
2. storage resources: global and performance transferring, capabilities management or enquiry mechanisms.
3. network resources: control over resources for enabling QoS, enquiry mechanisms, etc.
4. code repositories: software configuration management capabilities for source and object code.
5. catalogs: datasets for browsing and discovering of resources, configuration information for operations and management, etc.

Connectivity layer

The connectivity layer establishes the communication and authentication protocols which enable data exchange among different fabric resources. It is built upon fabric layer in order to provide the abstraction to resource layer which enable seamlessly accesses to fabric resources.

Communication requirements comprise transport, routing and naming capabilities. In most of the cases, this will be mapped to equivalent protocols in IP stack, but general use of resources, specially networks, involves the development of some mechanisms for adding extensibility and abstraction to that approach.

Regarding authentication requirements, this layer should provide the next features:

1. single sign on (SSO): ability to get authenticated only once (just entering into Grid services).
2. delegation: ability to endow a program to access services under user's behalf. This one together with SSO enable the integral security model required in Grid environments.
3. Integration with local security: seamless integration with local security solutions already deployed into some organization (such us Kerberos, Active Directory, Java Enterprise System, etc.)
4. User-based trust relationships: mechanisms for centralizing usage granting when services are provided among different providers. This avoids interactions between them as a prerequisite of use.

Resource layer

Resource layer protocols enable secure initiation, monitoring and control of operations over individual resources. Like in the analogy with IP stack, this layer establishes the constraints upon which upper layer protocols will interact with resources. So, this level performs its actions entirely over individual resources. This distinction is the key oppositeness to collective layer in terms of, for example, atomic issues (like the reliability property provided only by TCP in TCP/IP model).

Two classes of protocols can be considered:

1. information protocols: for enquiring about structure details or statuses like load or policy issues.
2. management protocols: for accessing and processes creation, monitoring and control. The important aspect is this layer serve as policy application point (so called policy enforcing point) in order to ensure consistency with the policy definition under which the resource is shared.

Collective layer

The collective layer is built upon the resource layer to extend its individual capabilities and provide global aware interactions across collection of resources hosted in different locations. Again, like in the TCP/IP stack equivalent (transport level), this layer can be considered the first end-to-end layer.

As a result of this design, provisioning properties mentioned before ⁷ are supplied by this layer. Some of these services are the following ones:

1. Directory services: discovering and enquiring of VOs properties or statuses.
2. Co-allocation, scheduling and brokering services: to request and establish distributed allocation of jobs.
3. Monitoring and diagnostic services: for supporting failure tolerance and improve delivered QoS.
4. Data replication services: for maximizing performance data transfers in term of latency, reliability and cost.

⁷The second key set of features in Grid environments after the virtualization of resources.

5. Grid-enabled programming services: for enabling high abstract programming languages to use global Grid services seamlessly.
6. Software discovery services: to discover best software implementations in term of platform requirements and user requirements.
7. Authorization services: acting as the policy definition point inside each VO (implementing RBAC security model).
8. Collaborative services: for coordinated exchange of information between large user communities.

Application layer

Application layer is the last level in the Grid protocols stack and comprises all the end-to-end user applications which interact with collective layer (also with lower layers) in order to provide all desired features. Applications are based on high level languages and frameworks that implement the mentioned services within every layer. So that is the mean for achieving interoperability and portability at user level.

1.3 Results: Grid potential

After presenting this overview about Grid technology, the adjoining question could be like *'what is the scope of this new computing revolution? why do we need something like Grid technology?'*. Further than philosophical implications or new cross integration of systems along different fields or sciences, Grid computing aims to improve global use of resources enabling the full integration of different powers in a seamlessly way [5]. The next sections are examples of target applications that Grid can improve significantly and thus, deploy all its powerfulness.

1.3.1 Distributed supercomputing applications

Distributed supercomputing applications represent the class of applications that require so high level of demands that they can be achieved only by joining multiple resources, becoming formally a virtual supercomputer. The main topics covered by this kind of applications are simulations (physical, medical, weather), real-time analysis (image rendering or signal processing) or complex analysis from very data-dense sources like telescopes or accelerators.

Supercomputing applications can be classified by the scale of their resource requirements in terms of peak or throughput speed, memory or data volume. And despite all of them share basically the same requirements, the method or model may vary from one to another. In some cases, heterogeneity can be considered as some added value because allows to integrate different solutions available to different platforms in a sole and unique way, avoiding additional efforts of, for instance, porting the applications to the available resources.

One kind of constraints supercomputing applications have to face up with is network issues, such us bandwidth or latency. Grid can contribute to improve the performance with the following characteristics:

Pipeline decomposition in applications that process from sequences of complex operations to series of data elements. It is just needed to identify and separate the pipeline stages to bind them to the proper resources.

Functional decomposition for applications in which functions can be decoupled and distributed across different resources with some minimum interactions.

Data-parallel decomposition for applications which apply the same algorithm to every data element, which is already loose decoupled with the rest.

Using these decompositions in the proper way for every case, applications can be developed for cases where performance depends exclusively on the underlying technology. Therefore, those problems like bandwidth or latency will evolve with the Grid and will be solved under the specific environments.

Another constraint is the ability to schedule the different components of a distributed supercomputer. This issue becomes crucial in order to obtain the desired peak performance in the application because with the proper scheduler, simultaneous access to the collection of resources in the right moment and using the right networks (which reduces again network bottlenecks) can be achieved. For these cases, the key functionality provided by Grid, currently unsupported in the supercomputer applications is the reservation mechanism, which allows this perfect orchestration to happen on time and during the needed time.

The last fundamental point offered by Grid technologies in supercomputing applications is the implicit fault recovery scenario. In traditional supercomputer applications, the time required to perform some job is quite long. To avoid having to rerun the entire job in case of failures, the checkpointing mechanism was invented to recover the application to some known

state (checkpoint). However, due to Grid distributed nature, usage of fault tolerant computational algorithms together with communication protocols allows either eliminate completely or reduce significantly the need to checkpoint in this kind of applications.

1.3.2 Real-time distributed instrumentation

Real-time distributed applications are the kind of distributed applications which have special requirements in term of response time to some event or data calculation. These applications are important in research environments because are used to control many of the precision instruments, which cannot be controlled by simply human interaction.

Usually, this time restriction comes from the management of high-speed data streams which provides the control information of the underlying instrument (e.g. health care systems, high-energy physics experiments or remote microscopy control are a few examples). Their management involves remote operation of the functions using a middleware supporting infrastructure (i.e. automated cataloging, storage interfaces, automated monitoring, policy-based access control, security, integrity, automated brokering or rich media capabilities) and of systems and communication services (like different transport schemas, reliable and unreliable multicast, reservation or network-level security).

Again, the nature of Grid computing enables exactly that behavior. The current design of Grid allows to implement every aspect of remote operation (locality, brokerage or time decoupled) using standard mechanisms for resource cataloging, reservation, allocation/brokering and security access.

1.3.3 Data intensive computing

The term data-intensive computing describes applications which evolve great amount of I/O operations, that is, an important part of execution time is devoted to movement of data. One criteria to identify such applications is by evaluating the total amount of data processed per CPU operation. However, computer memory usually acts as a cache for the real data source. So, another way of identifying this applications is comparing the required access rate to data stores.

But due to the great amount of data to get, these data stores show another important requirement: the data availability relies on mechanisms to identify and publish the proper data sets. In data intensive applications, if the data source is not high quality the results are suspect and it is possible good data sets will never be used.

Hence, these important requirements are fulfilled by Grid computing. It has all the key elements to provide these features by design:

data-handling are digital libraries which enable categorized access to data sets and data caching systems to provide access to different copies of the data without overloading the network capabilities.

information discovery performs data publication based on quality assurance mechanisms or information discovery interfaces which enable access to the specific data-handlers to get the right information source in every time, place, target (different metadata associated) and quantity.

knowledge networks represent multiple sets of expertise, information and knowledge that, aggregated, are able to analyze global problems (in terms of predictions or interpretations of existing data). This supposes to have published semantics and schemas and interoperability mechanisms.

Therefore, the Grid approach to data-intensive applications requirements provides systems to hold data (data storage system), systems to discover data sets (data naming system), mechanisms to retrieve and operate with the data (data-handling and data service systems) and high-quality data repositories (data publication and data presentation services).

1.4 Implementations

The computing Grids concept is a very abstract way of systems interconnection to some common purpose. As seen, the main goal is to design a versatile Service Oriented Architecture which enables richer interaction between own resources and contributed resources. However, the way in which this connection becomes, that is, the real systems architecture, can be optimized to focus on the different specific requirements or features from the seen in previous sections.

Currently there are many different Grid implementations, so this section introduces the most representative middleware in terms of architectural model, provided services and widely adoption.

1.4.1 The Globus Toolkit

The Globus toolkit is the first and most extended Grid-enabling technology. It was conceived at 1998 [3] as the way of sharing computing power, databases and other infrastructures across corporate, institutional and geographical

borders without sacrificing local control and autonomy. It was designed following the mentioned Grid principles: enable seamless collaboration between organizations by accessing remote resources as locally but preserving fully local control over its usage.

Specifically, Globus is a Service-Oriented Architecture which provides access to computing infrastructures based on web services referred as Globus containers [6]. These ones are application containers designed following international standards from OGF/GGF or OASIS when available and prepared to run Java, C or Python web applications. This one constitutes the general framework that enables the developers to browse, access and monitor the execution and storage environments, always covering security considerations.

The Globus toolkit is packaged as a set of independent components which provide the starting point to develop applications across the different layers (figure 1.1).

Common runtime components The common runtime components are the set of components intended to hide the complexity of dealing with service-oriented applications and service-oriented infrastructures. Specific details like message encoding, securing message exchange, service interface description, service discovering, etc. are implemented using the current standard technologies such as SOAP, WSDL, WS-Security, WS-Addressing, WS-Resource Framework, WS-Notification, WS-Interoperability, etc. In this way, developer has to care only about developing the service which performs the desired logic in one of the three supported programming languages.

This package provides C common libraries, C WS Core, Java WS Core, Python WS Core and the eXtended I/O ⁸.

Security The security component is known as Globus Security Interface. This component is based on the Public Key Infrastructures and SSL to provide the following features:

1. single and mutual authentication: relying on some set of trusted CAs, GSI can prove the identity of each part involve in some communication using the certificate mechanism.
2. encryption: by default GSI does not encrypt the communication between parties, but once the authentication is performed, it can be used to establish an encrypted communication using symmetric cryptography.

⁸C-written library for data transfer which supports multiple protocols as drivers.

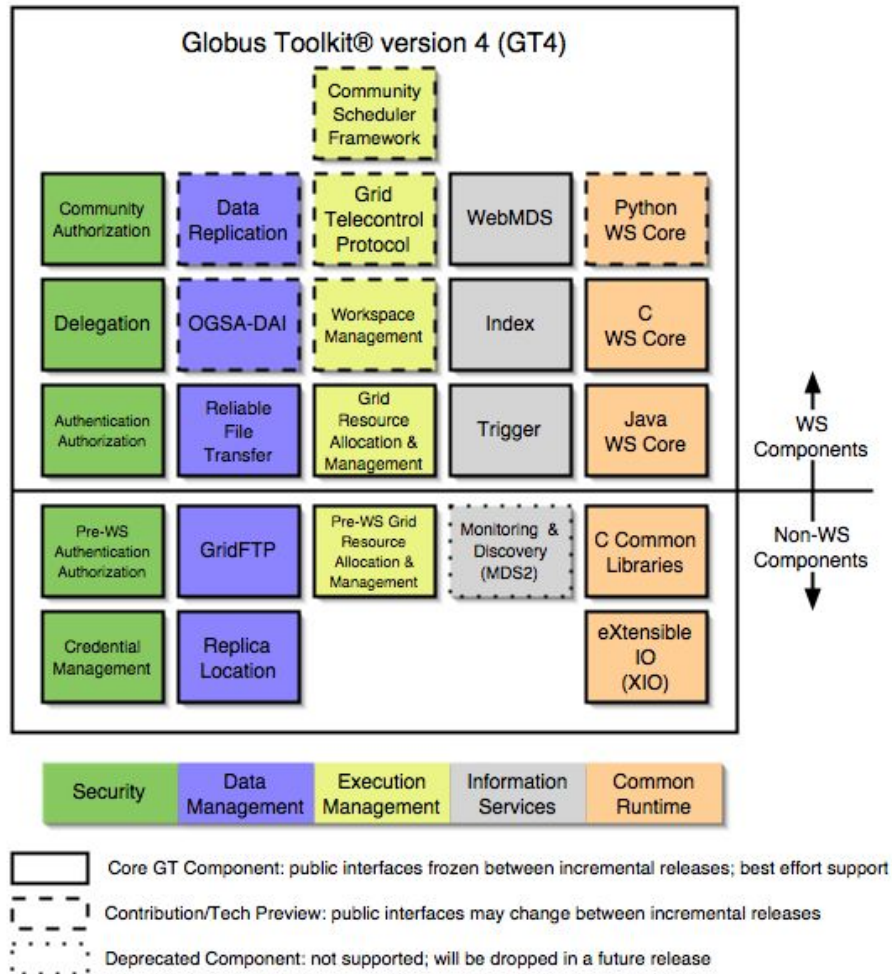


Figure 1.1: The Globus toolkit components

3. securing private keys: instead of storing the user private key in plain text, GSI enables the use of special containers (e.g. PKCS format) which require a passphrase to access the key before every new connection.
4. delegation and SSO: GSI enables temporary grants of personal rights to third parties to perform long time operations in the resources. This is known as proxy delegation and with this SSL extension, the user generates a short life-time certificate authorizing the system to act on his behalf during that time. As a consequence, once created, the user can use it to perform new mutual authentication without using his private key again.

As well as this common service, another services provided by this component are credential services like MyProxy⁹ and GSI-OpenSSH, the GSI-enabled version of the common OpenSSH.

Information services The Globus Information Services consist basically of the Monitoring and Discovering System (MDS). This is a suite of web services to discover what resources belongs to some Virtual Organization and monitor them.

MDS services provide mechanisms to retrieve detailed resource data from several sources and a trigger interface to perform specific actions when some condition is met. It includes two services:

1. Index services: collect data from different sources via query or subscription interfaces. Is based on WS-RF so the information is published as resource properties.
2. Trigger services: collect data and compare against set of predefined conditions in order to perform the required actions when there is any match.

Both of this services are based on an aggregator framework which unifies the data collection by defining common aggregation sources such as queries, subscription to services or calling external information providers like Hawkeye, Ganglia, WS-GRAM, RTF, etc.

⁹Usage of central security proxies has been discovered as the best way to ensure the integrity in the stored credentials under non-reliable scenarios.

Execution management The Globus Execution management provide a suite of services and web-services to submit, control and monitor processes in Grid computing resources. The execution management involves several aspects of the job execution: the job definition, the input operations for arguments or the output operations for results.

The execution management engine of Globus is called Grid Resource Allocation and Management (GRAM). It is intended to address the job management in terms of monitoring and control the overall job life cycle, the file transfer, staging in and out of computing resources and the credential management. In any case, GRAM is not a resource scheduler but the service which connects the local resource managers between them and any central load manager using a standard and interoperable message format.

Data management The data management tools are concerned with the location, transfer and management of the distributed data used during Grid computations. Globus Toolkit includes several tools and services to deliver high-performance and reliable data transport (GridFTP), to manage multiple data transfers (RFT), for maintaining location information for replicated files (RLS), and for accessing integrated structured and partially structured data (OGSA-DAI).

About data movement, the GridFTP covers the secureness, robustness and fastness in a transfer of bulk data. On the other hand, the Reliable File Transfer covers the fault tolerant aspect of the transfer implementing a WS-RF compliant web service that persists the transfer state in reliable storage, making the file transfer reliable itself.

Data replication services provide the ability to keep track of copies of files in the Grid. The Replica Location Service is a distributed registry that keeps track of where copies exist on physical storage systems (distinguishing between global logical filename and physical filename). The Data Replication Service provides a pull-based replication mechanism to ensure that some specified file exists on a storage site.

Finally, the Open Grid Services Architecture Data Access and Integration (OGSA-DAI) provides a way of querying, updating and delivering data via web services integrating different types of data resources (relational data, XML and files) without exposing them.

1.4.2 gLite Middleware

gLite is the service-based middleware architecture designed by the European-funded project EGEE. As a Grid architecture gLite is based on the SOA paradigm in order to provide the required abstraction layer among resources

and applications, aiming to facilitate interoperability between services by conformance to upcoming standards. It is influenced by the current requirements for Grid applications promoted by international forums like OGF and OGSA and by previous experiences like EU DataGrid (EDG), LHC Computing Grid (LCG), Virtual Data Toolkit (VDT), Globus Toolkit, Condor, NorduGrid and AliEn.

gLite has been specifically designed focused on the following set of requirements, some of them initially drawn by the OGSA: heterogeneous environment support, resource sharing across organizations, optimal resource utilisation, global job execution, efficient data services, security, low administrative cost, scalability, availability, application specific requirements for EGEE itself¹⁰.

In order to achieve this purpose, the middleware is structured in six main independent components [7] [8]. However the most important issue addressed by gLite covers the definition of scopes within the provision of the services, known as: user, site, VO and global. These scopes and the corresponding set of enforcing policies define the mechanism that allows virtualization of resources by user or communities, providing tools to guarantee the QoS subscribed by each one.

Security services provide mechanisms for authentication (based on trusted domains, revocation methods, credential storage and privacy preservation), authorization (based on Policy Enforcing Points such as the VO Membership Service and the Community Authorization Service and Policy Decision Points), auditing (for traceability of security operations and later analysis), delegation (usage of proxy certificates to grant set of privileges inside several entities to third parties and SSO identification) and sandboxing (isolation of applications to ensure user privacy and resource integrity).

Information and monitoring services provide the mechanisms to publish and consume information about resources or services in order to use it for monitoring, reservation or brokering purposes. The basic information services provide a low-level service to publish information about the status of specific resources. Upon this basic service, there are two specialised ones to perform richer operations decoupling the source of information from the consumer (i.e. the Job Monitoring Service and the Network Performance Monitoring).

¹⁰Inside the EGEE consortium, there are several partners representing different scientific communities such as biomedical, HEP or chemical ones.

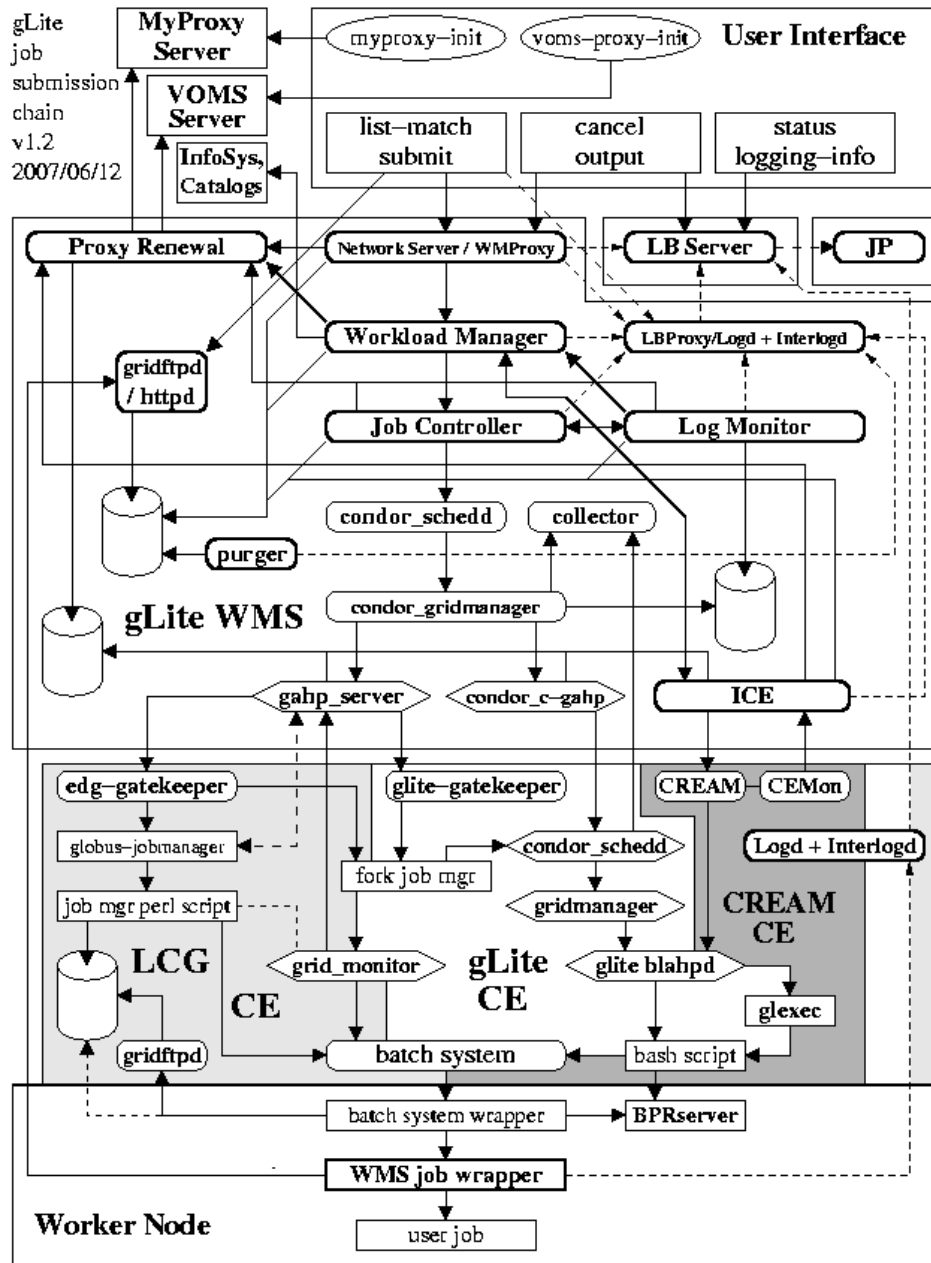


Figure 1.2: The gLite architecture

The security implications of this information service such as poisoning of caches or access to privileged information are solved using the scopes defined in the security component and enabling the control of published information only to the publisher.

Job management services provide all the functionality required to manage the whole life of any job. This task requires the following services:

- **accounting:** accumulates information about the usage of resources, enabling generation of reports and publishing information to track user activity. This information is intended to provide fine-grained usage reports and implement submission policies.
- **computing element:** represents the computing resource and its main functionality is to provide the service which control the job execution on each moment. This control involves several aspects of the job life cycle as the matchmaking process, the job description, the job control and the stage of input arguments, results and errors.
- **workload management:** is the responsible for the distribution and management of task across the Grid resources in terms of conveniency and efficiency. There are two main services, the Workload Manager System (scheduling and queuing) and the Logging and Bookkeeping Service (tracking the events of the job during its life cycle).
- **job provenance:** keeps a snapshot of the complete definition of the job, its environmental conditions and checkpoints to use this information later for debugging, post-mortem analysis, comparison on different scenarios or assisted re-execution purposes. But all data of completed jobs is stored.
- **package management:** automates the maintenance of software packages in a shared area. Operates in the context of a VO understanding and resolving possible dependencies between packages needed for the job execution. It constitutes an extension of the traditional Unix package management systems like RPM, deb, ports, etc.

Data management services provide the required elements to deal with the bulk of data information the resources use. The minimum granularity is on the file level so in this case, object-based access is not implemented.

In some way, the data services behave like conventional file systems. They provide logical file names in a hierarchical namespace (accessible from any

point in the Grid), unique identifiers (GUID like inodes in local file systems with mapping 1:1 with LFN), directories and symbolic links (no hard link concept).

Three basic categories classify the services offered:

- storage elements: are the last element where data is stored and the motivation of this component is to abstract the large variety of devices (with their QoS). Thus, the capabilities supplied are the next ones: storage space, resource management interface, space management, POSIX-like I/O access, file transfer requests.
- catalog services: store and deliver information about the data and metadata operated by the SE. This service is responsible for providing the file system appearance and resolving the physical location of the files. All the features are offered using several interfaces such as: authorization, metadata, replica, file cataloging, file authorization and indexes.
- data movement services: provide reliable and scalable capabilities to move data between Grid sites. The internal service decomposition is organized to schedule the transfers (Data Scheduler), arrange the transfer details (File Transfer/Placement Service), queue the request to get persistence (Transfer Queue) and finally perform the request between SE (Transfer Agent).

Helper services are intended to provide higher level abstraction in the management of the global Grid infrastructure and get better quality of service. The helper services are namely:

- Bandwidth Allocation and Reservation Service: is an end-to-end service to allow the fair usage of the network, balancing and shaping traffic and get the desired QoS delivered to the user.
- Configuration and Instrumentation Service: is the service that represents the set of operations and metadata related to set the state of applications in the desired state, that is, the mechanism which allows some central service to monitor and change the status of every application in the Grid.
- Agreement Service: implements the communication protocol used to exchange information about Service Level Agreements. This schema requires three main actors: Agreement Initiators (they trigger the negotiation to get the SLA desired), Agreement Service (enforce the

SLA guaranteed by the RASP) and the Reservation and Allocation Service Provider (performs the admission control of the resource under its control).

Grid access are the set of technologies which allow use of Grid resources. Grid access technologies constitute the final user which desires to perform some computing task, access to some information source, monitor the status of some job, administrate some features of the resources under his control, etc.

There are two ways of using the Grid. The former is using the available libraries which implement the interfaces defined by the services (APIs). This allows developers access to any layer in the protocol stack, depending on the level of performance or abstraction they need. The latter is using the Command Line Interface, which is usually an already developed tool wrapping the mentioned libraries.

1.4.3 UNICORE

UNICORE stands for *Uniform Interface to Computer Resources*. The initial goal was to develop the platform which enables *seamless, secure and intuitive access* to heterogeneous computing resources of German supercomputing centres. The first version was designed in 1997, parallelly to the becoming of Grid concept the new paradigm for distributed computing [9].

Despite being designed parallel to Grid, it can be considered other kind of Grid infrastructure because both share the same abstraction and provisioning concepts using a Service Oriented Architecture [10] [11]. In the case of UNICORE, its design principles are resource virtualization, security, site autonomy, ease of use and ease of installation. They are achieved with a three tier architecture (figure 1.3): user tier, server tier and target system tier. The current implementation of each level is made using Java as the programming language and following the Grid standards proposed by OGSA.

User tier is represented by the client interface. It is a pure Java GUI which exploits all the features provided by the underlying infrastructure. It consists on several modules to guide the user through the authentication mechanism (X.509 PKI based); the definition, submission and monitoring of tasks; the configuration of the sites with whom interact and contains also a plug-in framework to simplify the extensibility.

The connection with the server tier is performed sending Abstract Job Objects (AJOs). An AJO is a Java object that contains the full description of

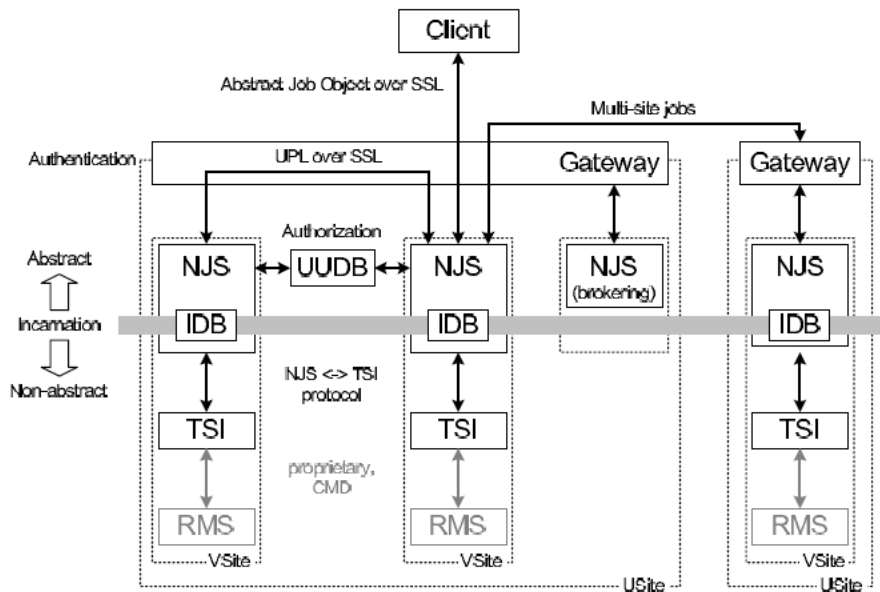


Figure 1.3: The UNICORE architecture

the task to perform in terms of computational and data resources, workflow process and security information but following a standard format that ensures platform and site independence.

Once an instance AJO is created, the client sends it to the server tier via the UNICORE Protocol Layer (UPL) tunneled over the SSL protocol. The information sent contains the object serialized, signed with the user credentials and some optional user data required during the execution.

Server tier is represented by two components: the Gateway and the Network Job Supervisor (NJS). The former is the entry point to the UNICORE service. It identifies by itself the hosted site (U-site), enabling to attach it to multiple Grids, authenticates the user and forward the incoming requests to the NJS of a virtual site (V-site).

A V-site groups a particular set of resources and is controlled by one NJS. Therefore the NJS acts as user mapper between global and local users (checking rights in the (UNICORE User Database - UUDB)). It maps abstract jobs on specific target systems (process called *incarnation*) using the Incarnation Database (IDB) and finally perform the stage of input and output files.

Target System Tier represents the interface to the underlying supercomputer infrastructure. It is just a stateless daemon which acts as the interface between the NJS and the Local Resource Manager System (typically PBS, LSF, Condor or Globus GRAM).

1.4.4 OGSA initiative

The Open Grid Services Architecture is the service-oriented architecture proposed by the Global Grid Forum (GGF) whose main purpose is to address the need of standardization so that Grid environments are able to speak together sharing interoperable, portable and reusable components and systems. The resulting architecture is based on the analysis of current requirements, challenges, use cases, previous experiences and the state of the art in related work.

The requirements stated by OGSA for standard Grid environments are based on a set of non-functional requirements that do not constitute the formal set of requirements but cover infrastructure and applications in scientific and commercial areas (i.e. service-based distributed queue processing, Grid workflow, Grid resource reseller, interactive Grids, etc.). The requirements enunciated are: interoperability and support for dynamic and heterogeneous environments, resource sharing across organizations, resource optimization, quality of service assurance, job execution, data services, security, administrative cost reduction, scalability, availability, extensibility and ease of use [12] [13].

According to this, the OGSA uses a building block approach to implement the services, their interfaces, the individual and collective state of resources belonging to these services and their interaction among them. The specific software implementation is outside OGSA efforts. Next, a brief explanation about each one of the capabilities proposed:

Infrastructure services are the set of common implementations, standards and knowledge that will be used across the rest of the infrastructure, some of them broadly used and others under design. This approach is based on the use of web services technology (XML, SOAP, WSDL, WS-I), naming schemes and policies, security mechanisms and policies (WS-Security, WS-Agreement, TLS, SAML, XACML), state representation (WS-RF, WS-Management family), notification and subscription (WS-Notification, WS-Eventing), atomic transactions (WS-TX, WS-Composite Application Framework), orchestration, OGSA profiles (defining the use of specifications for some specific purpose and ensure interoperability).

Execution management services are concerned with the issues of creating and managing units of work until completion. Some of the problems must be addressed are finding execution candidate locations and selecting them, preparing the execution environment, initiating and managing the execution, retrieval of results and decisions about interrupted operations. The solution adopted is to split the service into different classes: resources management services (model processing, storage, executables, etc.), job management services and resource selection services (to perform the matchmaking).

Data services focus on the management of, access to and update of data resources and the transfer among the sources. The specific requirements such as data movement, management of replicated copies, run queries and updates or federation of data sources are satisfied with a set of services which deal with different kind of data oriented to such different scenarios¹¹.

The functional capabilities provided by these services cover the next aspects: data transfer among various entities, storage management, simple access, query mechanisms, federation across sources, location management, update of contents, replica management, transformation (e.g. formats), security mapping extensions, resource and service configuration, metadata catalogues, data discovery, data provenance and global system properties (i.e. scalability, performance and availability).

Resource management services perform the management of resources in Grid. This management could involve the physical and logical resources by themselves, the OGSA Grid resources (using the service interfaces) or the OGSA Grid infrastructure (using the management interfaces). At the lowest level, the management demands to monitor, setup and control the resource and the native interfaces are used. The infrastructure level introduces a manageability model and interface which lets manipulation using web services. Finally, at the OGSA level two interfaces are defined: the functional one corresponds to services that perform already some kind of management by themselves and the manageability one is associated to each OGSA capability for its management.

Security services are intended to facilitate the enforcement of the security-related policy within a virtual organization. Its components must

¹¹The services are designed to deal with simple files, streams, DBMS, catalogues, derivations (intermediate result of any operation) or the own data services.

integrate and unify current security mechanisms, be decoupled from the implementation, easy to integrate and extensible to new approaches. The functional capabilities are: authentication (using several schemas), identity mapping (from the user authenticated credential to the suitable local one), authorization (using the proper mechanisms to attach the user request to the specific set of rights granted), credential conversion (i.e. between Kerberos systems and PKI-based systems), audit and secure logging (tracking all the relevant security events to later check compliance with access-control and authentication policies) and privacy (related with the policy-driven classification of personally identifiable information).

Self-management services perform the tasks related with self-configuring, self-healing and self-optimizing the IT infrastructures upon which Grid works. These processes are triggered in situations the service can detect by itself. The main objective of this kind of services is to support service-level attainment for a set of services so that the behavioral aspects of the component that cannot be determined a priori by the service developer can be achieved by a self-managing system using some set of business policies (Service Level Manager).

Those functional capabilities includes automatic adaption to changes in the environment, mechanisms that detect improper operations and initiate policy-based corrective actions or mechanisms able to tune themselves to achieve better efficiency according to business needs.

This comprises activities like service level management (monitoring, analysis and protection), policy and model-based management, resource entitlement, planning or provisioning in a security, highly available and high-performance environment.

Information services provide the ability to manipulate dynamic data and events used for monitoring, data used for discovering or data just logged about applications, resources and services. Typically, these services will produce information for execution management, accounting, resource reservation, resource usage or monitoring services.

The functional capabilities provided by these services are: service and resource discovery (involving directories or registries, indexes, peer-to-peer discovery or multicast), message delivery infrastructure (using a message broker and providing reliability), logging and monitoring.

1.5 Summary

Starting from the previous status and pointing the new challenges, this chapter presents the Grid paradigm as the future proposal that aims to give answer to the upcoming requirements of larger computing power.

Regarding the energy power grid, the first authors outlined the way of applying the same concept to the computing resources, envisioning a new way of producing and consuming infrastructures along all the world. This new approach requires to fix the issues that arise from a very abstract and versatile structure where two main principles shape the proposal: virtualization and service provisioning.

Currently, the theoretical effort has been followed by the efforts from different communities to adapt this generic solution to several scientific disciplines. After several years of development and proofs of concept, it has been demonstrated that the approach is feasible so the first distributions appear. There are many different versions but the first one, the Globus toolkit, depicts the basic building blocks that could be used to create more complex designs.

In any case, this spread of efforts to achieve physical results needs to be concreted in software components that, delivering adequate quality levels, can be used universally. There are several initiatives, like the OGSA, to put together efforts towards these objectives and only upon successful accomplishment of their activities, the Grid will become what expected.

Next chapters will cover a specific subset of these tasks, the job submission standardization through the OGSA-BES recommendation.

Chapter 2

State of the Art

During more than thirty years, the computer scientific community is contributing to the software engineering development and its quality process in many different ways. One of the efforts is oriented to fully understand the complex environment which surrounds the software creation process (i.e. its difficulties and its common troubles) trying to identify techniques and common patterns to give answer to it. Concretely, this complexity is devised having a look on the great amount of activities where any software component is used, from common information systems to specific drivers to control critical installations or biological activities. Moreover, apart from the implicit difficulty of problems to be solved, it can be said that other component of the complexity is identified as the extensive range of software applications.

In average, there are 20 to 50 errors per 1000 lines of code during the development phase and 1.5 to 4 per 1000 during the maintenance phase [14]. This shows that testing procedures are very important, not only to reduce the number of errors but also to improve the efficiency in such task.

The first part of the chapter will cover precisely this, which ones are the processes involved in the delivering of a good quality software, the trends, standards, phases, and methodologies. In a second part, the system testing phase will be presented in more detail, showing its common challenges and its crucial role in distributed environments in general and in Grid in particular. This will introduce the concept of testbed as the most extended approach to address the troubles drawn. Finally, the third part presents some of the existing initiatives trying to cover the topic.

2.1 The software quality process

In software engineering literature, there are many efforts to set what quality means. This great diversity mostly resides in the trials to fully understand the quite recent, very complex and full of implications task of designing working software. But despite all this variety, there is a common agreement in what can be considered as a quality software. IEEE states that quality is *the degree to which a system, component or process meets (1) specified requirements or (2) customer or user needs or expectations*[15]. This definition covers the two most common perspectives of software as a project, that is, systems that satisfy consumer perception and systems that satisfy the three producer constraints, namely: time, budget and specifications.

The great amount of research performed on this topic has identified the establishment of the standards-based mechanism common in the general industry as the way of guarantee to the user that the product is of a certain quality. But in opposite to these environments, in software industry, the abstract nature of the object under development makes quite difficult the definition of an universal standard to achieve quality in the final release. Currently, there are several standards to cover almost all the phases involved in the software development process. Some of them overlaps or contain what appears to be contradictory requirements, but they agree in the next set of principles:

1. the whole development process is driven by plans defined according to standards
2. conformance with standards quality requirements needs to be measured
3. verification and validation are integral part of the software quality assurance

The first principle concerns about the project infrastructure, how it is managed, how the plans are designed, how standards are implemented, which activities are carried out and how (software development, documentation, testing), policies in case of failure, etc. [16]. The second principle focuses on defining the set of general parameters that allow to get a quantitative state of the software, the meaningful values for such parameters and the way of measure them. The third principle refers to the processes designed and performed to check the software against the specification requirements and the user requirements. These tasks consist of technical reviews, walkthroughs, software inspections, checking user and software requirements traceability [17] and testing.

The complexity and human expertise requirements of the first principle make it to be out of the reach of this thesis. More information can be found on [18]. For the rest of principles, the central topic is software testing, but metrics are closely related to it so next a brief introduction to them is presented.

2.1.1 Quality metrics

Quality requirements of software products are often described in vague and general terms. This means that is quite difficult for test engineers to evaluate the quality of such products because there is no unambiguous or quantitative reference. Such problem is not only present in software engineering world, but also in any industrial process that deals with manufacture of products.

Aiming to address all those issues, several organizations have developed their own guidelines for quality assurance. The most important and recognized ones are the family of standards for quality assurance developed by the International Organization for Standardization (ISO) and the Institute of Electrical and Electronic Engineers (IEEE). The former defines the ISO 9126 for *Software Engineering - Product Quality*, which customizes the recommendations from the general ISO 9000 for *Quality management and quality assurance standards* (focused any industrial process) to software development. The latter defines the IEEE 730 *Standard for Software Quality Assurance Plans* and the IEEE 1012 *Standard for Software Verification and Validation*.

The quality model defined by ISO 9126 allows to measure the quality of software products by classifying quality properties into six categories, namely: functionality, reliability, usability, efficiency, maintainability and portability. Although not all these characteristics are of equal importance to a software product, this schema provides the base coordinate system where to check the evolution of a product in a quantitative manner. Usually, this constitutes the starting point to the quality management plan needed to filter and deduce the list of important parameters that apply to the underlying product. Creating a quality profile enables to get focused on the specific requirements and allows to define the metrics that will be used. The standard also proposes two kind of metrics called external and internal ones.

The definition of a metric implies to define the procedures and tests that generate numeric data but also to define the reference units that will be used to compare with the obtained data. This is the most difficult task because defining the processes may be as difficult as defining the value that provides and acceptable quality. Also, the analysis of such values should be interpreted with great care since the weight of some characteristics must be

taken into account. Hence, this shows that an experienced based quantitative evaluation is required.

Therefore, despite the completion criteria are not met and the tension between time-to-market and product quality always remains, the final goal of metrics is to provide a method that supplies a major input for management to identify risks and support the release decision process.

2.1.2 Software testing

Software testing is the task in software engineering that covers the verification and validation activities, that is, checks the software against its designing specifications and the user requirements.

IEEE defines verification *as the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase*. IEEE also defines validation *as the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements*. Consequently, both activities establish the purpose of testing by covering two different but related scopes: on one side, to check that the code resulting from the software development process performs in the way it was designed and on the other side, that the design itself resolves the user problem¹.

Demonstration that a product fully meets its specification is a mechanical activity driven by different specification requirement documents and is efficient for demonstrating conformance to functional requirements. In contrast, demonstration of no defects is a non-trivial task that usually requires expert knowledge of what the system must do and the technology used. Nevertheless, both aspects are automatable using the wide knowledge on this field taking different available tools.

Validation is a more complex task. The proper operation of the software has already been checked so success depends on the final user perception, which is not so easily automatable. This requires most of the times interaction between experts and the users themselves. Consequently, the rest of this chapter will focus on testing from the verification point of view, showing the most common methodologies, phases and existing tools.

Phases

The V-model (figure 2.1) is one of the most common approaches to the development life cycle. Its structure incorporates testing phases to the traditional Waterfall development process. Its structure is also an agreed way

¹As an idiom says: ‘Do the system right and do the right system’.

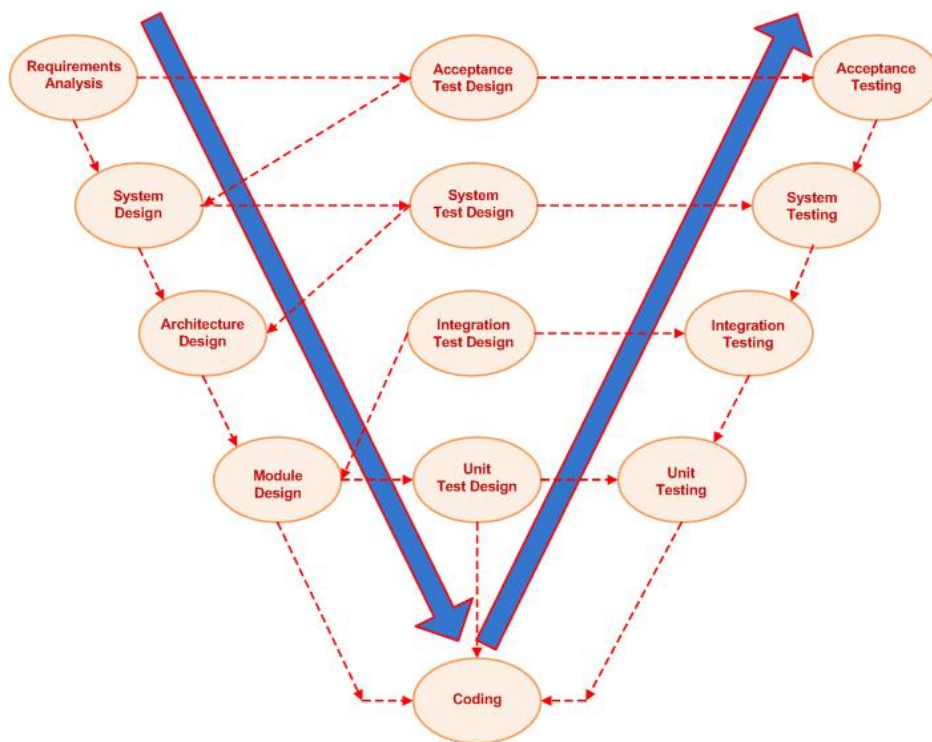


Figure 2.1: The V model of software development

of building software for any kind of project where several development teams are involved, specially in distributed environments because it easily decouple roles in the release process.

The V-model defines four testing phases covering the testing of results produced by every design phase. These four stages group different kind of software testing methodologies, each one applying to the specific concerns of the design phase.

Unit testing refers to the process of testing modules against the detailed design. Its inputs are successfully compiled modules from code and are assembled during the process to obtain larger units. The test is designed to check the low level functionality provided by the component and may be aware of the internal implementation (white-box unit tests) or unaware (black-box unit tests). There are also two different strategies when executing the test. Considering the module to test has several components, the bottom-up approach checks first the independent components using drivers to simulate the higher-level components and the top-down approach checks first the dependent components using stubs to simulate the lower-level components.

Integration testing refers to the process of testing subsystems (compounded of components) considering their contribution to the global architecture. The objective of this phase is to expose defects in the interfaces and the interactions between the components or systems considered.

System testing refers to the process of testing the full system against its software requirements, thus, the input must be the successfully integrated system. The scope of system testing is to verify compliance with the system objectives. Its designs must consider details of the software requirements and identify the test use cases and proper procedures to performed it. Possible approaches are function tests:

- functional tests: to check functional requirements
- performance tests: to check quantitatively the system performance
- interface tests: to verify conformance to external interface requirements
- operations tests: all the tests that check user interface, man-machine interface or human computer interaction requirements (e.g. learning curve, usability, response time)
- resource tests: to check that system requirements such as CPU time, storage space or memory are accomplished and not more resources are used

- security tests: to check that security mechanisms are in place to avoid confidentiality, integrity or availability voids
- portability tests: to discover coding defects or platform specific dependencies that avoid to run the software in different platforms
- reliability tests: to check the Mean Time Between Failure (MTBF) of the software, considering different degrees of failure (e.g. critical, non-critical)
- maintainability tests: to define the Mean Time To Repair (MTTR) of the software (i.e. the time between reporting of a software problem and retrieving the repair)
- safety tests: to detect software failures that may cause a critical or catastrophic hazard and that must be avoided at any price
- regression tests: to retest a system or a component to verify that newer modifications have not caused unintended effects (important before every release to prove that the capabilities of earlier releases are unchanged)
- stress tests: to evaluate a system or software component beyond the limits of its requirements (i.e. maximum concurrent requests of data processed simultaneously)

Acceptance testing Acceptance testing is the last process of the testing in the V-model and corresponds to the validation part against user requirements before mentioned. Its input is the software successfully tested at the system level.

In some practical cases, the V-model is considered a theoretical approach because does not fit properly with specific industry requirement. However, most of the alternatives perform the same steps but in a different order or at a different rate, what makes it to keep its validity. Some of these software development processes are: spiral model, chaos model, formal methods, iterative and incremental development, clean room, rapid application development or agile processes.

Methods

It has been presented the test phases that are usually performed in some or other way during the life cycle of many project. The importance of test planning is as large as the test execution itself, but it is not always implemented in the proper way because of lack of time or resources. Planning the testing is even the first test of the specifications and lets to solve development

problems before they appear. So in order to accomplish this task, there are several different methods, focusing on various scopes of the test phases. Next, some of them are described [19].

Category-partitioning helps converting the natural software specification to a formal test specification. This method is carried out in five steps:

1. Identification of the minimum functional units that cannot be split into smaller ones.
2. Creation of a set of categories that represent high-level attributes of the inputs.
3. Partition the categories into specific values or value-ranges that can be assigned to the input (e.g. choices that force a default setting or behavior to occur or choices in the boundaries).
4. Constraints determination among the choices of different categories (e.g. setting impossible combinations or mandatory ones).
5. Generation of test frames from test specifications (i.e. combination of categories, choices and rules) that are considered general test cases. From here, real test cases are just obtained by customizing these test frames with real values.

This method has some advantages like to make a test specification instead of test cases, which is more flexible; test cases are reduced, so it is less probable to leave out important cases; to know what test cases are left out, having a clear idea about the quality; or the learning curve is reduced.

Syntax testing is a black-box data-drive testing technique for applications where input data can be formally described. It is based upon analysis of the component input syntax in order to model its behavior via its input. Using a context-free language, sentences are generated and injected into the component to check if it accepts, rejects or fails. Therefore, the implicit advantage is the possibility of automation.

Some domains where this technique can be applied are: GUI applications, markup languages parsers, CLI commands, database query languages or compilers.

Equivalence partitioning is a technique that classifies all the possible input and output values of a component in equivalence partitions. This compression of values cuts down the number of test cases used to test the system reasonably. The disadvantage is that being heuristic-based,

not all the inputs are tested. Also, the definition of the equivalence class may be not that simple.

Boundary value analysis is commonly used together with equivalence partitioning and consists in the selection of the values at the extreme ends of and just outside the ends of the input domain when testing any functionality. Both limits, upper and lower, must be derived from the specification of the component. It has the advantage of revealing easily interface input problems generating a small set of test cases. However, neither all possible inputs nor dependencies between them are tested.

Error guessing is often linked to equivalence partitioning and boundary value analysis. It is a test design technique bound to the experience of the tester. It involves making list of errors that are expected to occur and then designing the test cases to check them. The great disadvantage of this technique is that is hardly repeatable.

Cause effect graphing is used to derive test cases from a given natural language specification to validate its corresponding implementation. It models causes as conditions and the effects as actions to be performed. The boolean graph generated relates inputs and outputs with boolean operators. It was used to point out ambiguities or incompleteness but difficulties on its maintenance make it old fashioned.

State transition diagrams models the component behavior defining states, their transitions, the events which cause them and the actions that result. Thus, this technique involves analysis of the transitions between application states, the events that trigger them and the results. The better the model captures the specification, the most effectiveness is achieved. Once this is reached, the recommended approach is to test every state-transition combination at least once.

Random testing consists in generating test cases randomly over the entire input domain, according to a predefined distribution (e.g. normal, uniform). Such distribution is based on the expected operational inputs range and when no clue about it, the uniform distribution shall be used. Despite being able to be automatable, the quality of the test set may be poor.

Fuzz input testing generates test inputs randomly by applying a huge character string containing both valid and invalid characters. Depending on the kind of application (event driven, character driven or database), the input data will consist of a queue of data structures,

streams of random data or random data filling the standard database schema. Consequently, the purpose of this technique is to demonstrate that a software component handles exceptions without crashing, rather than behaving correctly.

Smoke testing is a non-exhaustive software testing technique that check the proper behavior of the most critical functions but without getting into finer details. It is intended to be performed after build tasks and just before the system tests to confirm changes do not break the build.

Equivalence testing is a technique that checks the correctness of a software implementation by comparing the output with another standard implementation for the same input. The disadvantage is that two pieces of equivalent software are needed.

Tools

The panorama depicted until now shows what kind of concepts and strategies are involved in the verification and validation of software. Because of its abstraction, some of them are arduous or simply expensive to implement in a tool. Next, some of the contributions to this topic from the global community are described.

CatGen is a tool for automating the generation of test frames from formal test specifications according to the category partition technique. The input is a test specification, written in scripting language, and the output consists of the test frames together with the extended information of generation. It may also be used to create a test hierarchy, by comparing different specification versions of some test.

The benefits of using CatGen are the automation of non-intelligent part of category-partition technique, flexibility of test frames, usable in almost every type of software or testing phase.

Mercury TestDirector is a web-based application for managing the whole testing process. It gathers requirements, plans and schedules tests, analyses results and manages defects and issues. It consists of five major components:

1. Requirements management: links test cases to application functional requirements (ensuring traceability)
2. Test plan management: is the base to design the test plan and build test cases and centrally stores the information enabling sharing across projects

3. Test lab management: schedules all the test to run unattended and enables dependencies among tests to simulate business processes
4. Defects/issues management: controls the entire defects life cycle, covering all the steps passed on their resolution and avoiding duplicates
5. Reporting: collects all the information generated during any of the mentioned testing processes

Log4J is an open source framework for logging Java applications. It is intended as a first aid to developers, speeding up the debug process. By itself is not a testing tool rather than a supporting tool for tests, complementing the information provided during some test execution.

It defines hierarchical levels of logging (i.e. fatal, error, warning, info and debug) and log statements which are returned at arbitrarily fine granularity, using the mentioned hierarchy and inheritance of loggers. Hence, it allows control over which logging statements are enabled (using loggers), manage the output destinations (using appenders) and the format of such output (using layouts). The only drawback is the lack of asynchronous logging.

xUnit is a unit and regression testing framework architecture for several programming languages. Its objective is to provide a framework where developers can produce their own tests while the development process happens. Another goal is to create set of tests that remain in the time and that can be integrated across different developments without interference.

Essentially, xUnit provide the mechanism to execute tests or suites of them automatically, setting the testing scenario and disabling it after the test. There are a few reporting options to generate XML/HTML outputs. The most common variants of this unit testing architecture are: JUnit (Java), PyUnit (Python), accessUnit (MS Access), AS2Unit (Action Script 2), CFUnit (ColdFusion), MinUnit (Minimal C), CUnit (C), cppUnit (C++), dyUnit (Dylan), lUnit (Lua), dotUnit (.NET).

GJTester is an implementation of the Computer-Aided Software Testing (CAST) paradigm for unit and component testing in Java. It is comprised of two major components TestEngine and TestEditor. TestEngine executes the functions and procedures defined in the test scripts. TestEditor is the script editor, provided by a friendly GUI. Its basic role is to enable fast input test creation by showing all the possible options to

add the script instead of writing source code or using difficult syntaxes. Another role of the editor is to monitor the execution and report the results in a comprehensible format.

The basic units of the test script are `TestCall` and `InstanceCreator` that can be grouped together into a test case and a set of these ones in a test suite. The subject of a `TestCall` is a concrete instance of a class which contains the function under test.

JTest is a unit testing and code analyzer tool for Java. It checks whether the code complies with more than 500 Java development rules and code conventions. Rules can also be customized without coding. JTest can examine classes to expose reliability problems, achieve high coverage and expose uncaught run time exceptions. It also includes a Test Case Sniffer to monitor running applications and generate JUnit test cases which capture the application behavior.

Indus is a framework for static analysis and slicing of concurrent Java applications. Slicing uses program statement dependence information to identify parts of a program that influence or are influenced by some set of program of interest. It consists of three major components, Indus, `StaticAnalyses` and `JavaProgramSlicer`. The former contains the common interface definition to perform analysis and transformations on the system components. The second one collects the different static analysis and provides the framework to check value and object flows, thread escaping and dependencies.

JavaCov is a coverage tool for Java whose purpose is to measure the effectiveness of unit test cases. It supports two types of coverage, branch and multi-conditional decision coverage (MC/DC).

Branch coverage is a test method to ensure that each possible branch at some decision point is executed at least once, therefore ensuring that all reachable code is executed. It can detect all the Java control flow statements and also the *throw* statement (although not included in the statistics). MD/DC is a refined version of the branch coverage whose criteria is to require that each condition must be shown to affect the outcome of such decision independently.

It supports two test case types, as Java application and as JUnit, but in any case it does not support the design of test cases, only gauge of the coverage of such tests. Also, the monitored program must be instrumented before running the test cases (what is automatically done using the tool itself).

Mercury WinRunner is a functional and regression testing tool which captures verifies and replays user interactions automatically seeking for defects. It runs test cases written in TSL language. These test cases use to be macros that represent the set of operations to perform. It can use checkpoints to compare expected and actual outcomes from the execution. Another option is to group atomic test cases into test scenarios, useful for regression testing.

Some of the key features are the test recording capability, grouping test into scenarios and the possibility to test GUI, web or database-based applications.

vTest is a functional and testing framework made to verify and validate web applications in a variety of environments. It supports test creation by recording interactions with web applications. When recording is finished, a script is generated which can be edited, compiled and run again. During the new execution, a detailed HTML report is generated containing elapsed times, the iterations performed, statuses, the pages visited with the loading times and different checkpoints. As can be seen, the great benefit resides on the lack of programming, generating data-driven tests.

ITP is a test tool for functional, regression and smoke tests of web applications. I can be used for test-scripting linking building blocks and generating the corresponding script in XML format.

Apache JMeter is a Java load and performance testing tool. It can be used to test performance both on static and dynamic resources by simulating heavy load on a server or network elements. The execution procedure is based on a test plan describing the steps to be executed (i.e. generating controllers, listeners, timers, assertions or configuration elements).

Despite it can be used to perform distributed testing, there are some drawbacks like no subnetting support, is easy to overload the controlling console because of the extensive amount of results and is a quite resource-consuming tool.

WSRP is a test kit that allows to check conformance with the OASIS WSRP specification. It includes a monitor for collecting web service messages and an analyzer to process the log files, determine conformance and generate the corresponding reports in XML format.

PushToTest's TestMaker is a web application and web service testing tool designed to perform functional and stress tests. It provides a GUI

to generate tests from a WSDL interface and in scripting language, a test object-oriented library for the creation of web services tests (using different protocol handlers). It also provides a network console for distributed testing and a wizard to write scripts (Agent Recorder).

2.2 Description of a testbed

Until now, theoretical aspects of testing as part of software engineering have been introduced, describing why of their origin, their requirements and most common techniques as well as guidelines to integrate them into the whole development process. Nevertheless, nothing has been said about the environment that surrounds such process, the main actor which orchestrates the activity and how all such methodologies influence in the development of any product since the beginning. This one is precisely the concept covered by testbeds.

An important part of references to testbeds in technical literature [5] [20] mention it as the virtual scenario used to model the desired set of features from some future project (i.e. new technology or product). Moreover, this model represents not only the use cases definition of the product but also the proof of concept in terms of feasibility, inherent troubles, further implications or even trials for creating an additional interest in form of community around the new concept.

Particularly, the IEEE stand will be used here again which defines a testbed as *the environment containing the hardware, instrumentation, simulators, software tools and other support elements needed to conduct a test*. In this definition the key point is the environment context in which the tests will be performed. Defining an environment means to set the elements that create a fully-controlled reproduction of a real scenario. This one isolates the test from production constraints, dependencies and risks when running it on a production one, but keeping the features required like in one of them.

Therefore, added to this definition, three important aspects of a testbed are the next:

- hardware: the set of platforms or physical infrastructures that provide the computing, storage and network capabilities that enable any kind of work and interaction between the components of an hypothetical software development under test.
- software: the set of supporting software applications that interface the hardware infrastructure with the global functionality provided by the testbed (i.e. configuring, monitoring and control tools). In this context,

it provides the control and instrumentation interface to the underlying resources.

- **simulators:** the set of specific applications that exploit the powerfulness of the virtual infrastructure and deploy and run the software and test cases under consideration.

As can be noticed, this testbed description shows an important analogy with the description of the Grid infrastructure made in the section 1.2. In fact, it uses the same virtualization and provisioning principles stated in the Grid solution for the different purpose of controlled and reproducible test making. This coincidence naturally suggests that the Grid ability to perform some tasks under certain conditions could be used to extend the functionality and capabilities offered by conventional testbeds technologies in case of being Grid-based.

A particular case of interest for using testbeds is the possibility of reproducing under a controlled and systematically way (i.e. programmatically) some complex or rich-interactive scenario where the distribution of components, their integration itself or the interoperability between different implementations of services are the target of the test. This is usually covered by the system testing phase of the V-model but it deserves a special mention to know exactly what it involves.

2.2.1 Distributed testing

Despite it depends on the specific purpose, distributed systems are basically designed to satisfy a common specific range of features such as reliability, availability or performance. This high-end capabilities involve problems such as service discovering, synchronization, data consistency and data replication, concurrency issues, fault tolerance, security, etc. So, in addition to the usual testing procedures to check the validity of software, specific tests that check the functionality under actual conditions are needed.

Distributed testing refers to this exercise of verifying and validating the features of some application which implements the distribution paradigm [21]. Thus, this exercise usually covers deployment of software in a remote scenario and check the simultaneity and synchronization among processes by checking how the interactions happen.

As mentioned before, distributed testing is close to system testing in terms of scope and target systems. The methodologies applied to the latter are still valid for the former, but this new scenario involves the necessity of new techniques to cover special concerns of distributed testing like portability or interoperability.

In general, the execution of a batch of test cases in a distributed system will involve:

- **definition:** detailed information about the structure of the services participating in the test (i.e. scope, functionality and dependencies), the information they will exchange and the method in which it will be performed
- **orchestration:** definition of how the distributed system is going to be launched, defining the timeline of the service deployment and the intermediate results needed by other parts or components
- **deployment:** sequence of steps which enables the automatic installation, configuration and running of every service involved in the test
- **synchronization:** mechanisms that avoid races or blocks between components based on the information of the system definition, orchestration and deployment
- **execution:** ability to launch, monitor and reset the set of test cases upon the underlying system, acting as main actor who set the environment (using the previous stages) run the tests and collect the results

2.3 Existing solutions

The panorama shown until now has described the concepts about testing in general and distributed testing in particular, illustrating which are the principles, the approaches and some tools. The broadness of these activities is not so easy to implement in the day-to-day management and usually requires many efforts that not all the companies can afford. In big corporations, this issue is often addressed either developing their own infrastructure or outsourcing the service provision to specialized suppliers.

In addition, the increasing complexity of the projects developed by the open source community forces to implement some kind of distributed organization that leverages the available resources, keeps the flexibility of such structures but also includes the powerfulness of standards and recognized best practices.

In order to cover this void of developing and testing tools for big project, some initiatives have appeared trying to address the common troubles and improve the quality delivered by such communities. This section shows the state of the art in terms of tools for distributed testing and user target.

2.3.1 The Apache Software Foundation

The Apache Software Foundation has sponsored a number of projects to develop common tools and standards for software construction. It specifically provides different tools to assist the developer during the development process. However, these projects are often referred to as *social experiments*, since they try to address not only purely technical problems related to the software development management, but also the issues that arise when software is developed by different communities in geographically apart locations. It promotes many different projects for software developments but next two ones have special interest for the purpose of this document.

Gump

Apache Gump is a Python application aimed to serve as continuous integration tool. It supports natively Apache Ant, Apache Maven and other building tools to build and compile software against the latest versions of projects. Therefore, the main goal is to perform build and tests every few hours to detect possible incompatibility issues between different versions of project components. In this case, according to the previous description about software testing, it can be regarded as a building and testing tool because it integrates the continuously developed code seeking build-time errors.

It works by defining the project structure in XML format. This file contains all the information required to build the project in terms of sources of code, dependencies with other projects and external tools needed. With this information, Gump analyzes the dependencies, uses the proper VCS command to checkout the code and run then compilation, piping the output of the dependencies to the requirements to the next project component.

The results are brought together in a HTML-format page, showing successful operations, failures and the environment variables like program versions used during the whole process. It is also able to send build reports to group of users via e-mail.

Maven

Apache Maven is designed to provide a clear definition of what a Java project consists of, a standard way to build it and an easy way to publish such information as well as resulting libraries across different projects².

Since the primary goal is to enable fast comprehension of the state of a development, Maven provides an easy and uniform build system, quality

²It defines itself as *a software project management and comprehension tool*.

project information, guidelines for best practices development and transparent migration for new features. For instance, the quality project information is delivered by generating change log documents directly from the source control, cross reference sources, mailing lists, dependency list or unit test reports. The best practices are taken into account by setting the specification, execution and unit test reporting as part of the normal build cycle. It also maintains the test source code in a separate but parallel source tree, uses test cases naming conventions, etc.

Hence, as a project management tool, it features simple project setup following recognized best practices, consistent usage across projects, dependency management including automatic updating or dependency closures, extensibility, release management and distribution publication among others.

2.3.2 Sourceforge

SourceForge.net is the world's largest open source software development site. It provides free hosting services to about 100.000 open source projects focusing on easing the creation of solutions within collaborative environments. The tools provided by SourceForge.net are oriented in two directions, on one side to improve the communications within development teams and between developers and user. On the other side, managing the tools and services that make possible the development and release processes.

The Collaborative Development System (CDS) is the main tool which allows project administrators to fully manage the project (i.e. select and configure services, manage lists of team members and their roles, manage the release and publicity system, etc.). The most important services provided by SourceForge.net are:

- VCS services: CVS and SVN repositories are enabled to store the code generated by the projects
- Communication tools: web-based trackers (for easy management of bugs reports, support requests, features requests and user-submitted source code patches), web-based forums (to host discussions between developers and between developers and users about development or usage issues) and mailing lists (to provide subscription mechanisms for information updates to different parts).
- File release tools: to make available ready-to-use software packages to the end user. These tools provide the full procedure to release a software component without concern about the maintenance of the

middleware infrastructure (i.e. the globally distributed network of download mirrors).

- **Publicity:** to increase the visibility of the project in the global SourceForge.net repository. This includes tools like keyword-based searches; specific details about concerns, scope or family of products; monthly publicity in the selected *Project of the month*; higher positions in the top page based on the project statistics (downloads, last activity, etc.); project announcements via syndication feeds (e.g. RSS) and shown in the SourceForge.net front page; screenshot system to provide software previews to potential users.
- **Management:** facilities for managing the available storage, access to a shell to enable the generation and maintenance of project web content, web capabilities (based on different programming languages such as PHP, Perl or Python), manage the DNS name space for the project providing different 3rd level domains.
- **Donations:** tools to channel the supporting action of the user community in terms of economic contributions or hardware resources.

2.3.3 Metronome

Metronome is the framework developed by the University of Wisconsin-Madison for building and testing software in a heterogeneous, multi-user, distributed computing environment. It is part of the NSF Middleware Initiative for creating the build-and-test facility that can be comprised of a few computing resources in a single location or a large heterogeneous collection of machines spread geographically.

Its main motivation is precisely to abstract the underlying infrastructure and processes in order to encapsulate them in separate and fully automated tasks. Thus, users can migrate their existing tools to the new facility without compromising the already deployed procedures. In the abstraction process, the system is able to safely execute routines on resources outside the local administrative domain (therefore not assuming systems configured exactly the same). Due to dealing with this heterogeneous environment, the system also provides resilience by allowing to restart any running activity in other resource when the former eventually becomes unavailable. Moreover, to achieve this, Metronome uses mechanisms to describe the capabilities, constraints and properties of the resources in the pool it manages. The system schedules the routines to be performed on the resources that match

the user requirements, deferring such execution until the machines become ready.

The design principles of the tools are the next:

Tool independence The tool does not require any specific build or testing tools. Users are provided with a general interface that allows them to encapsulate their own application-specific scripts in well-defined building blocks processed by the framework.

Lightweight The framework and its related tools are designed to be small and portable. This provides easiness for all the set of operations (administrator and user sides) and access to external resources without need of installing complex additional packages. It runs on the top of Condor, using its workload management and distributed computing features.

Fully controlled environment All the external dependencies and resource requirements must be explicitly defined (either already present on the target machine or installed by the framework). This ensures predictability, reproducibility and reliability of results. This sets the pre-requirement to create and isolate the proper execution environment on demand.

Central results repository All information data generated by the tasks (execution results, logs and program outputs) is automatically gathered in a central repository. This information can be used later for statistical analysis of the builds or tests.

Fault tolerance The framework is resilient to error and faults of the underlying components by using the capabilities provided by Condor. If the worker node cannot contact with the submitter node, the results are collected locally to be sent back when the communication is restored. On the other side, if the submitter node cannot contact with the worker node after some defined lease, the task is restarted in other different resource.

Platform independence Despite the system is designed to perform tasks only on the matched resources, in case of multi-platform applications there is also the possibility to define platform-independent tasks that will be run only once.

Build and test separation The output of any task can be used as the input of another one. When needed, the system will recover from the

central repository and deploy on the target machine such results. This allows the user to decouple targets clearly by their scope and compose them according to the workflow requirements.

Furthermore this design principles, the main Metronome characteristic is the ability to provide the additional logic on the top of a heterogeneous environment (the Condor batch system) to run safely simple jobs in a Grid-like scenario and control the whole workflow (i.e. fetch, pre-processing, platform and post-processing phases).

2.3.4 Others

CruiseControl

CruiseControl is an open source framework for managing the continuous integration build process. It consists of three main components:

- *the build loop* is the core module. It triggers the build cycles and notifies various listeners using different mechanisms (extensible via plugin framework). The trigger can be internal (changes in the software configuration manager) or external (via predefined configurations).
- *the legacy reporting* is the publishing component. It enables users to access the log outputs and artifacts resulting from the build task. This results are generated in XML format and presented to the user in HTML format by applying the user-defined XSL transformation and CSS style.
- *the dashboard* is the information tool which enables visualization of the current project status. It also shows the relevant information using a color code for the success and time of the performed jobs. In this manner, the user can get easily a fast overview of the project.

Mozilla Tinderbox

Tinderbox is another open source project to quickly check the current status of some development project. It cover specifically four mains aspects of the development:

1. *automatic build and test*: it performs the tasks to launch and monitor the continuously compilations and automatic unit test and code coverage metrics. It also monitors which commitments in the code cause it to fail and the logs of the process to check where the build breaks.

2. *last changes*: it shows the history of recent changes stored in the version control system. The idea is to show easily which parts of the code are being changed and who is making such changes. It offers the possibility to lock code trees in such way that later breaking commits to the VCS will be traced to the responsible.
3. *ticket tracking*: it shows the recent changes in the code tree as a result of bug fixing or new user requests.
4. *communication*: it improves the communication between developers and managers by setting the channels to announce changes, operations or simply information about the project status.

ElectricCloud

ElectricCloud is a commercial company focused on development of software production management. Its main branch of products is oriented to create the build and test facility which allows the user to control the full release process.

ElectricCommander is its most basic product which is focused on automating the software production process by automating the building, packaging, testing, and deploying aspects of the development. Some features of this product are ability to manage any kind of project without regarding its size; sharing and reuse of project assets, splitting the complex project structure in different related building blocks; making processes transparent and repeatable and easiness of learn and adapt to changing environments, allowing to reuse the already deployed structure without re-engineering.

ElectricAccelerator is a solution intended to reduce the bottleneck present in the most of build jobs by parallelizing builds across scalable sets of resources. It analyzes jobs and their dependencies to spread out the build task over a pool of inexpensive machines and monitor the different processes to avoid broken builds.

ElectricInsight is an add-on to the ElectricAccelerator tool that visualizes the build structure for error detection and showing performance troubles during the parallelization. It keeps track of the information generated by ElectricAccelerator to present a full description of the build in terms of which jobs are executed, when and which ones are the relationships between the different parts.

BuildForge

IBM Rational Build Forge is the IBM product for automatic build and test. The main features are:

- automation of the build process by organizing repetitive tasks and enabling consistent processes
- fully compatible with existing procedures such as build scripts or deployment tools, what leverage the existing resources
- enabling multi-threading processing of jobs across a pool of servers and dynamic management with multiple scheduling options
- customizable monitors and loggers for the output processing
- audit trails to trace the full build activities
- exporting options to ensure reproducibility across different scenarios
- reporting and metrics capabilities to execute and display the tests performed
- web-based management and integration with the most leading IDEs (IBM Rational Application Developer, Microsoft Visual Studio and Eclipse)
- supporting many different platforms (distributed resources or mainframes) and OSs
- extensible by using public APIs for Java and Perl

2.4 Summary

Software engineering is the branch within computer science that cares about the creation and maintenance of computer programs to be efficient and accurate to their needs. One important part of it deals with the development of techniques and procedures to apply the conventional processes used in the rest of industrial sector to the software in order to assure the final quality of the developments. As part of it, testing is the specific phase that aims to ensure that developments fulfill the requirements in terms of correctness and accuracy, processes called verification and validation.

In order to verify that a piece of software is properly designed, it can be analyzed and measured according several standard guidelines publicly

recognized as good-practices. For that there are many different techniques and tools that aim to address smaller parts of the picture. Tools like Cat-Gen, xUnit or Cobertura implement different parts but always focused on analysis of the code (lowest part of the V-Model), but not on analysis of the functionality.

However, the quality assurance process also requires procedures related with the quality perceived by the end user, conformance to formal specifications and sustainability plans. But this topic is not so easy to manage, specially inside big projects where lot of developers contribute to the final artefact. In this cases, topics like distributed testing or the release management broadly determine the success of the final result.

In the community, specially the open source one. There are several alternatives available to aid the project to manage such complexity. The Apache Software Foundation with Gump and Maven, public organizations like the NSF with Metronome and private companies like ElectricCloud offer complete frameworks to arrange in several ways the build and test process. Despite none of them cover every requirement, all depict the desired features and experience in way of different approaches that a general solution must provide. This scenario sets the background where the ETICS project will develop its activity.

Chapter 3

The ETICS system

The ETICS project (eInfrastructure for Testing, Integration and Configuration of Software) is a European-funded project chaired by CERN whose main purpose is to support collaborative efforts in research and development initiatives. Such efforts are usually oriented to produce large-scale systems where a lot of different resources are implied. For this purpose, ETICS integrates existing procedures and tools in a coherent infrastructure in order to create the facility that enables the mentioned research projects to integrate their code, validate it against standard guidelines, test it extensively and run benchmarks and produce output artifacts and reports. The final outcome is a software component which satisfies the current standards in terms of interoperability and quality assurance.

Along this chapter, ETICS design principles, features and architecture are described, drawing the set of capabilities that will be used in the compliance test use case later on.

3.1 What is ETICS?

In many open source projects there is a common background problem, how to set up the software life cycle management to guarantee quality, interoperability and maintainability in the final artifact. This affects to the resource capability, human resources management or time and budget constraints in such way that the unsuitable management becomes in a low quality software (i.e. difficult to deploy, integrate and maintain). In large-scale projects, the risks are even larger because some factors such as developers spread geographically, with different programming languages and different platforms (and so tools) lead to a difficult environment to manage.

In the case of Grid developments, there are so many different services,

developed in collaborations between different teams, that the lack of some tool to deal with the software development cycle makes the final goal hard to accomplish.

The vision of ETICS is precisely to leverage the power of the Grid infrastructure already deployed today and help addressing the software development and maintenance challenges mentioned so far. ETICS aims to collect and settle all the information, tools and processes required and already in place usually in the projects resources in order to set up a reference common infrastructure. This will simplify the infrastructural, technical and cost problems of projects developing software for the Grid or other distributed environments and therefore, will improve the overall quality of the output.

3.1.1 Objectives

To fulfill this vision, the ETICS project aims to achieve the following sets of objectives:

Management support sums the efforts driven to create the powerful distributed facility that leverages the Grid capabilities developed by many of the projects to whom ETICS is primarily intended and lets to decouple the daily project management from the purely project base infrastructure management. In order to achieve this, ETICS purposes are the next ones:

1. To identify a number of existing pools of resources with adequate computing and networking capabilities which are susceptible to be federated in to a Grid way that will comprise the ETICS infrastructure.
2. Based on the analysis of a suitable number of successful software development projects, to identify their requirements in terms of supported platforms, programming languages, libraries and tools that will provide versatility to the intended ETICS infrastructure.
3. To identify the minimum project metadata and current build and test tools that included in ETICS will satisfy the identified requirements for distributed developments.

Quality Certification process sums the initiatives leaded to formulate and implement a common software engineering practices which let ETICS users to be automatically compliant with them, avoiding the setting up process. The final goal is to establish the mechanism and

tools to assure software quality, making possible the conformance with the certification process out-of-the-box. Specific objectives are the following ones:

1. To promote the definition and adoption of common configuration management and quality assurance guidelines which foster software reliability and interoperability.
2. To collect software configuration information and test suites used in successful projects and integrate them uniformly producing build, test and quality reports.

Promotion and dissemination of results sums the efforts driven to create the reference point which allows to store the status of the project and its results in order to improve the collaboration among different projects, industrial applications, extension to other communities, etc. Specifically, there are two objectives to achieve this:

1. To create the repository of software configuration information, documentation, benchmarking data and reference test cases used to generate some kind of software component.
2. To create the dissemination channels that connected to the repository allow to know to any developer or potential user a fully detailed description of the process followed by some software product. Elements such as test use cases passed, quality metrics, compliance matrices, trend analysis, etc. will be susceptible to be published by these means.

3.1.2 Features

According to the objectives outlined before, the ETICS system is designed to integrate seamlessly the information describing the project to be used during build and test stages.

Management support features

- Out-of-the-box automatic build and test system
- Multiplatform support: Windows, Unix/Linux
- Designed to support the current implementations for Version Control Systems

- Flexible automatic dependency management (based on internal and external dependencies)
- Possible to build from sources or use pre-build binary packages
- Extensible by a plugin framework mechanism to increase the build functionality

Quality certification process

- Trigger coding convention checks, unit tests, metrics generation, documentation-harvesting tools and publish the results
- Ability to virtualize real scenarios during testing by deploying automatically the different services on different machines
- Designed to guarantee reproducibility of results
- Production of different package formats
- Extensible by a plugin framework mechanism to extend the test capabilities

Dissemination of results

- Publishes rich build and test reports
- Automatic collection of information about results such as run-time information, run-time dependencies, etc. and publication in the repository
- Possible to register the artifacts in the repository in several ways
- Customization of the access to artifacts, enabling several package distribution systems (e.g. apt or yum).

3.2 ETICS architecture

Until now, ETICS has been introduced from the point of view of the user, showing the design principles, objectives and advantages that a potential user will experience since the first use. In this section, a deeper view of ETICS will show how those requirements are met: how the project information flows inside the system and which are the ETICS components that using such information reach the outlined features.

As mentioned before, ETICS is not a development project, that is, it is not intended to design any new tool to, for instance, to build or test software in a certain way. However, it establishes in a programmatically way, the abstract structure where to define procedures or dataflows that are common to many distributed software engineering projects and which use already existent tools to perform the real tasks in the project.

Thus, the description of the ETICS architecture is split in two major concepts: the data model which defines how the user data is gathered and the service architecture which defines the elements involved in the processing of such data.

3.2.1 Data model

For any tool which aims to deliver high-end automation levels or to behave as an intelligent system, the key point is what information the system holds and how structures it, that is, which kind of relationships can be inferred from data to allow the system exploit it and get the desired behavior. Discover the proper set of relationships as well as the latter classification in a generic hierarchy created ad hoc is what determines the degree of robustness, easiness or suitability to different nature projects the system can provide.

In terms of software development projects, ETICS identifies two main entities which structure any project: project modules and project configurations. Project modules gives anatomical structure to the project. This concept comes from OOP and represents the software elements that provide certain functionality, e.g. some authentication library, a web service, a system test, etc. However, these modules do not contain information about any specific version, repository of sources or dependencies. This specific information is related with some concrete version of the module considered. So here appears naturally the second concept entitled by ETICS as project configuration: the entity which provides physiological structure, defining the specific details of the module considered (such as source code repository and version, dependencies with other modules or configurations, etc.).

These ones are the key concepts in the ETICS data model and both are used to provide the build and test facilities. Next, each one is explained in more detail.

Modules

As mentioned, a ETICS module represents the generic container that stores information about some software element and relates it to the rest of project elements. Considering an analogy between ETICS and Concurrent Versions

Systems (e.g. CVS), ETICS modules have the same entity as CVS modules. The table 3.1 shows the minimum piece of information the user must introduce in the system in order to define the module. It is easy to see those fields do not provide any kind of information about which version of the element is being considered, how can it be built or tested.

For ETICS, any project can be settled defining its structure as a hierarchical tree of modules. However, the module concept is an abstraction of information container. Actually, ETICS distinguishes three kind of modules depending on their scope, namely: project, subsystem and component.

project is the top of the hierarchy and represent the software development project itself. As the root of the tree, all the parameters defined on it are automatically inherited by every element under it. It is compounded of subsystems and components. The project contains the information shown in the table 3.2.

subsystem is a container of software components where to apply specific properties and policies. The subsystem can also be a software component in its own to be used, for instance, as meta-package producer. One subsystem belongs to one and only one project and contains the information shown in the table 3.3.

component represent the smallest unit of functionality in the project architecture. The component belongs to one and only one project or individual subsystem and normally is used to generate software packages. It contains the information referred in the table 3.4.

deployment represents a deployment test to be used in the testbed environment¹.

node represents a service node to be used in the testbed environment¹.

service represents a service deployment module to be linked to a node¹.

test represent a test module to be executed as part of the distributed test¹.

All this information defines the skeleton of the project: general description, owner, corporate web site, components structure or even the default place from where either checkout the source code, either get the binaries without performing any additional operation. However, it is possible to see that there is no information about how to perform specific operations like for instance, build any component, bind the operations to some specific platform

¹A more detailed explanation can be found in section 5.1.2.

Parameter	Description	Mandatory
name	The name of the module considered (must be unique in ETICS)	yes
displayName	The friendly name of the module	no
description	A description about the module	no
repository	The default URL of the repository storing the packages of this module	no
vcsroot	The default root folder of the version control system storing source code for this module	no
vendor	The module owner	no
modifyDate	The date of last modification	no
createDate	The date of the creation	no

Table 3.1: Module parameters

Parameter	Description	Mandatory
<i>module parameters of table 3.1</i>		
homepage	The home page of the project	no
logo	The URL of the project logo (usable in a web environment)	no

Table 3.2: Project parameters

Parameter	Description	Mandatory
<i>module parameters of table 3.1</i>		

Table 3.3: Subsystem parameters

or launch a test scenario to execute some specific test. This other kind of information is more specific for the current status of the development and fully addressed in the second entity mentioned before: the configuration.

Configurations

An ETICS configuration groups the set of parameters that identify a complete and unique status of an ETICS module, that is, one module version. Considering again the CVS analogy, the homologous entity of an ETICS configuration is the CVS tag or the CVS branch. Both identify a concrete state in the software development process.

In the ETICS system, configurations are the objects on which the build and test operations are performed. When a module configuration is build, by default all child configurations on its tree are automatically built.

Each ETICS configuration can only be attached to one ETICS module, but each module can have more than one configuration attached. This forces to establish the relationships between configurations explicitly (instead of implicitly as with the modules) using the concept of subconfiguration. With this one, it is possible to get a complete snapshot of the project in terms of the versions used for each one of its components.

Any configuration, either project one, subsystem one or component one, behaves as a container of the same kind of information which is classified according to its scope as:

general parameters present in any configuration and listed in the table 3.5. In addition any configuration inherits the parameters listed in the table 3.6 from its corresponding module.

platforms define the environment under which this information is applicable.

specific commands refers to the fields needed to perform the specific build and test actions in one or several platforms.

dependencies refers to the resolution of the module dependencies. These dependencies can be either internal dependencies (i.e. dependency with another component of the same project) either external dependencies (i.e. dependency with other component belonging to some external project).

Parameter	Description	Mandatory
<i>module parameters of table 3.1</i>		
packageName	The name of the package produced by this component if different from the component name	no
homepage	The home page of the project	no
download	The URL of the download page for this component if taken directly in binary format from the vendor	no
licenseType	The license agreement to which the component is bound	no

Table 3.4: Component parameters

Parameter	Description	Mandatory
name	The name of the configuration (must be unique in ETICS)	yes
displayName	The friendly name of the configuration	no
description	A description about the configuration	no
age	The configuration age number	no
majorVersion	The configuration major version number	no
minorVersion	The configuration minor version number	no
revisionVersion	The configuration revision version number	no
path	The path where the package file can be downloaded from (relative to the repository)	no
profile	A comma-separated list of predefined profiles to apply to the configuration	no
status	The configuration status (alpha, beta, RC, etc.)	no
tag	The tag string of this configuration in the Version Control System	no
modifyDate	The date of last modification	no
createDate	The date of the creation	no

Table 3.5: Configuration parameters

Parameter	Description
moduleName	The name of the parent module
moduleDescription	The description of the parent module
licenseType	The license agreement of the parent module
projectName	The name of the project to which the parent module belongs

Table 3.6: Inherited module parameters

Platforms

Software development projects may have a strong dependency on the environment they are built. This dependency comes from the programming language, the available libraries or the specific features provided by some hardware architecture that makes the software more suitable for such environment. In addition, other projects can be platform independent because the technology used is like that (i.e. Java applications). ETICS supports this heterogeneity by introducing an abstraction layer which unifies the management of the underlying infrastructure and covers these differences. The concept of platform represents exactly this idea, the tuple consisting of hardware architecture, operating system and native compiler (i.e. the C compiler version). This structure creates a unique identifier for every development platform class.

Thus, the platform concept is the glue between the configuration definition and the specific commands that govern the available component operations. So every configuration consists of one or more platform definitions, each one defining the specific commands to be used under such platform. For the cases where there is no specific platform, a default platform defines the commands that will be valid for any of them. The platform matchmaking will be resolved by the client in execution time and depending on the user preferences.

Specific commands

In order to perform any build or test task, any automatic tool needs to know about how to get the code, how to use it and what to do with the results. In the case of ETICS, these questions are addressed filling specific data sets organized depending on the state they are used. In this manner, to tell the system about the commands to get the code to perform the task (build

or test), ETICS defines the VCS commands detailed in the table 3.7. The manner ETICS will use this code depends whether it is a build or test task. In the first case, the user must set the Build Commands (detailed in the table 3.8) to show the system how to build his code from the beginning to the packaging. In the second one, the Test Commands (table 3.9) define the steps required to launch the test in the Implementation Under Test (IUT).

In addition to these set of commands, ETICS provide two mechanisms to set additional information that may simplify the definition process. The first one is setting information as properties. These ones are key/value pairs that once established in some configuration, are automatically inherited by any subconfiguration. ETICS also defines a set of built-in properties according to the data contained in the module definition.

The second mechanism is based on environment variables. These ones are key/value pairs that are available in the execution environment while processing the task².

Dependencies

Normally, software projects and specially open source ones do not develop all the desired functionality from scratch but they reuse code provided by third parties. This behavior facilitates the development of new capabilities but creates dependencies between components that must be properly managed. ETICS aims to provide this management in a versatile way and for that it defines two categories of dependencies:

static are fixed dependencies between two configurations. They are specified by name and are not affected by properties or policies.

dynamic are defined between an specific configuration and a module. The specific configuration of the module is dynamically resolved at checkout time using properties and may change if the same parent configuration is used in different configuration trees.

In addition, both kind of dependencies have scope, depending on the task considered:

built-time are used during build operations but are not added as dependencies in the distribution package (i.e. need in compilation time).

run-time are used to define dependencies in the distribution package explicitly (i.e. must be present in final installation).

²Available in environments that support this mechanism (Unix, Windows, etc.).

Command name	Description	Mandatory
description	The command set description	no
tag	Command to tag code	no
commit	Command to commit code	no
checkout	Command to check out code	no
branch	Command to branch	no

Table 3.7: VCS Command parameters

Command name	Description	Mandatory
description	The command set description	no
checkstyle	Command to perform code checking operations	no
clean	Command to clean	no
init	Command to perform initialization operations that depend on the build system structure	no
configure	Command perform initialization operations that do not depend on the build system structure	no
compile	Command to compile code	no
doc	Command to generate documentation	no
test	Command to perform static tests in the code	no
install	Command to install the package	no
packaging	Command to generate distribution packages	no
prepublish	Commands to be executed before publishing the build artifacts	no
publish	Command to publish build artifacts to standard distribution formats	no
postpublish	Commands to be executed after publishing the build artifacts	no

Table 3.8: Build Command targets

Security model

The great amount of sensible information stored in ETICS and the big impact in case of abuse make ETICS to implement a strong authentication and authorization schema. The security model is based on two extended and well-known technologies such as Public Key Infrastructures (PKIs) for the authentication and the Role-Based Access Control (RBAC) for the authorization.

A PKI is the set of entities and procedures that allows to rely on the identity of some subject by checking the credentials provided (Certificate) against a third trusted party (Certification Authority). This technology is based on asymmetric cryptography so once the credentials are accepted, there is absolute certainty that such credentials belong to the individual (have not been tampered) and that the identity contained matches with the issuer's one so the subject is authenticated. The great advantage of this mechanism is that there is no prior and explicit action required by the user to be authenticated, the system trusts on every credential signed by the recognised CAs. In this case, ETICS trusts the European Grid Policy Management Authority (EUGridPMA).

The RBAC is the set of user roles and policies that allows to describe what every user can do. Instead of granting specific rights to every user, these ones are grouped into roles so the rights are granted to group of individuals. This mechanism eases significantly the user management and the security constraints enforcement. In the ETICS system, read-only operations are publicly accessible, but editing operations require the users to have one or more roles (detailed in table 3.10).

3.2.2 Service architecture

The data model presented until now shows how ETICS understands the structure and the relationships of a software development project. The fact this information can be established is a milestone itself in the project definition because eases and clarifies the objectives and relationships among all the parties involved during the development. But for ETICS, to achieve this definition means that it is possible to keep track of the evolution of the states since the beginning. The service architecture of the ETICS system is precisely oriented to that: once there is a project definition seed, this leverages the use of the system logic to initialize the environment, saving costs and allowing the developer to focus on the development.

One of the objectives ETICS aims to provide is the reproducibility of results as a mechanism to achieve and keep a good quality level of the

Command name	Description	Mandatory
description	The command set description	no
clean	Command to clean	no
init	Command to perform initialization operations	no
test	Command to execute the tests	no

Table 3.9: Test Command parameters

Role name	Allowed operations
Administrator	All operations on all projects (used by ETICS service managers)
Module administrator	Module edition and security management
Developer	Configuration edition and remote build submission
Integrator	Configuration edition, remote build submission and build artifact registering in repository
Tester	Remote test submission and test artifact registering in repository
Release Manager	Lock configuration, remote submission and artifact registering in repository
Guest	Read-only operations

Table 3.10: Roles

software. Using the previous data model, reproducibility means that once the project is defined, if the relationships among components (i.e. dependencies) are properly set the same results must be achieved across the time and in any supported platform. The first constraint requires some long term locking mechanism for the data so relies on the business logic. But the second one defines how the access from every platform to the service shall be performed, drawing therefore the service architecture itself.

Another important aspect considered in the design of the ETICS service architecture is the quantity of different tools used during the development process (i.e. building artifacts, unit testing, integration testing, system testing and acceptance testing) and the different scenarios where are used. The system not only must be able to provide the logical structure for the data management but also must give access easily to this data and integrate it seamlessly with the infrastructure. In this sense, ETICS considers two scenarios: local and remote scenario.

The former represents the operations that are performed locally under the direct supervision of the user. Operations such as build, unit testing, debugging or initial configuration setup are inside its scope and are associated with a primitive state in the development. In the other side, the remote scenario represents wider operations in the project such as the integration tests (i.e. white/black box testing, performance testing), system tests (i.e. security testing, portability testing, regression testing, etc.) and the acceptance test, covering the full quality assurance process. But despite of these scenarios, the steps needed in every one are essentially the same: fetch some code, deploy it and evaluate some function over it.

Thus the reproducibility of results mentioned before can be extended to the reproducibility of tasks. And this is what the ETICS service architecture basically provides: the mechanism to use the project data and the available resources to perform the tasks corresponding to each state in the software development process in a seamless and programmatically manner.

The ETICS service is designed as a 4-tier architecture (figure 3.1). The decomposition in layers splits the functionality of every application and service and also improves the scalability, manageability and resource allocation that the system provides globally. Having four levels makes possible to isolate clearly the functions provided and also abstracts the provider and consumer. Something that makes the system flexible to future extensions.

It is important to notice the important duality present in the figure 3.1 between the two different scenarios. In both of them the architecture is essentially the same: the user through the application layer connects with business tier to perform some task. This tier provides the business logic and access to the physical resources through the access tier. The difference

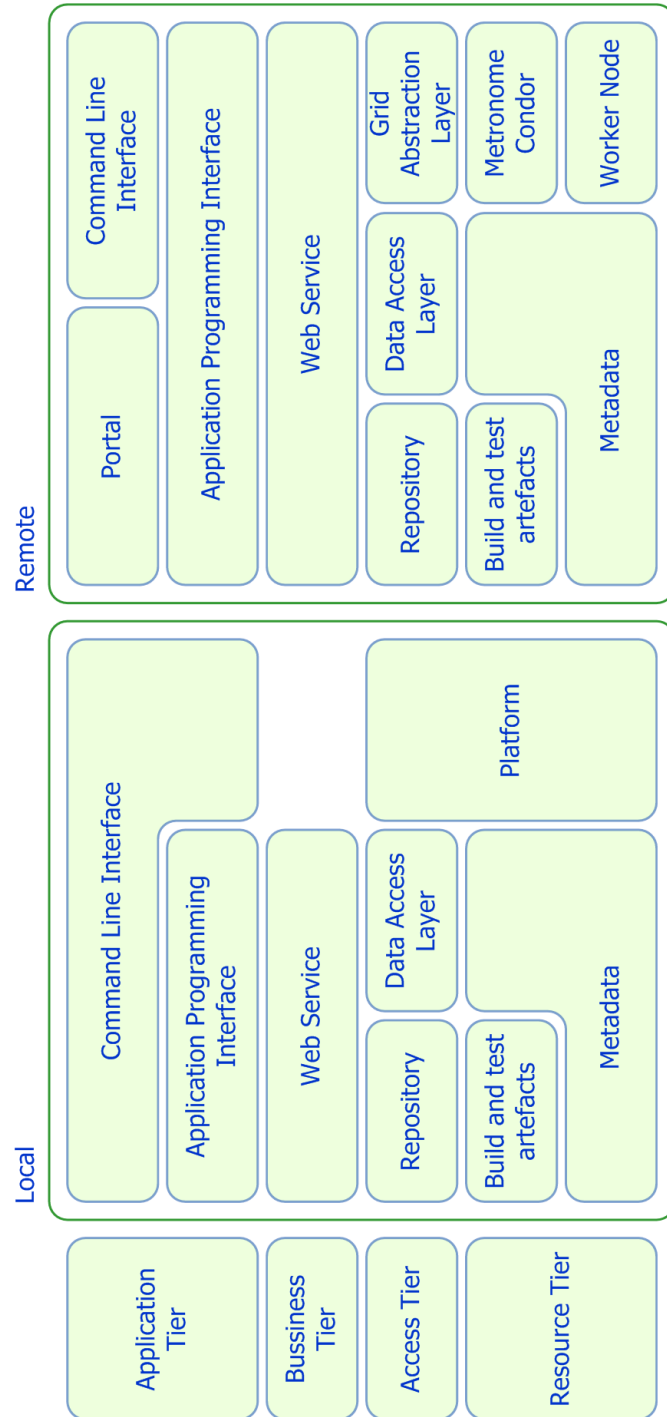


Figure 3.1: The ETICS Architecture

is in the way the computational resource is accessed. In the local scenario, the user through the command line interface accesses directly to the platform capabilities (i.e. running java compilations or pyUnit tests). There is no more logic in between that system libraries. However, in the remote scenario, the user delegates his role to the business logic. This one accesses to the resources using the underlying Grid technology (which selects the target platform) but, and here is the key point, in exactly the same way as the user would do: running the local scenario in the remote environment and getting the results back. This structure guarantee that what the user get locally, will be obtained remotely, but saving the management time.

Next, the functionality of each tier is explained with greater detail.

Application tier

The application layer is the semantic interface between the user and the ETICS system. Its mission is to be the exchange point between them, providing the mechanisms and logic to make the interaction rich but simple and translating the knowledge between both parts.

The application tier consists of three elements, namely the command line interface (CLI), the web portal and the application programming interface (API) and is mainly executed on the user side.

The CLI is the main user tool to interact with the system. It allows to perform the full set of features ETICS provides: create and manage modules and configurations, perform local and remote build and test operations, perform local and remote distributed tests (coscheduled tests in ETICS terminology), check the status of remote tasks and manage output artifacts and releases. Its behavior slightly changes from the local scenario to the remote one. The creation and edition of information is performed in the same way in both scenarios. The difference comes from the build and test operations. In the local scenario, the CLI is responsible for executing the sequence of tasks in the local machine where it runs, but in the remote scenario delegates such execution to other CLI launched in the target platform selected by the ETICS system.

The Portal is the graphical user tool to work exclusively in a remote scenario without requiring a local machine where to execute the CLI. It consists of three components: the Build and Test Web Application, the Administration WA and the Repository WA.

The Build and Test WA allows the user to create and manage the information related with modules and configurations as well as launch remote jobs on target machines.

The Administration WA allows the project administrators to grant rights to the people related in the project already registered in order to perform the specific operations drawn in the table 3.10.

The Repository WA allows access to the output artifacts from the system such as build reports, test reports and the packages generated.

The API is the set of libraries and utilities used by the client interface (CLI or Portal) to connect to the central service. For instance the stub generated from the WSDL file to connect to the Web Service.

Business tier

The business tier is the core layer of the ETICS system. It provides the business logic to gather the user information, to store it and to retrieve it answering incoming requests. To implement this layer, ETICS uses the Web Service paradigm. It allows to decouple the application tier implementation from the service implementation, what gives enough flexibility in the service provision.

As said before, the Web Service makes the core operations for the resource management. These operations cover the security enforcing point, the creation, modification and deletion of modules and configurations, the linkage between those ones, retrieval of the full or partial project information tree and also the registration and access to build and test artifacts in the information systems. The operations rely on the data provided by the information systems in lower layer and which give access to the physical resources.

The service announce its set of operations using the standard mechanism for Web Services, via the Web Services Description Language (WSDL). This document is publicly accessible to any potential user and is also implemented by the API in the application tier to create the client libraries.

Access tier

The access layer is the intermediate logic between the business logic and the physical resources. Its mission is to provide the interface to access generic resources, decoupling the real implementation of those ones. Specifically, the abstraction is made over two different kind of resources, data sources and computing resources.

Because in terms of data model there are two different types of data, raw data and the metadata, the access to each one will be slightly different. In this sense, the access tier has two components, the repository logic and the data access layer. The first one groups the set of technologies to store and retrieve the metadata gathered by the ETICS system during the whole configuration, build and test processes. The second one provides access to the raw data independently of the technology used for its storage.

In terms of computing resources, ETICS performs the build and test operations in the platforms describer earlier, but running on the final platform depends on the selected scenario. In case of local operations, the own CLI uses the system library to access the machine capabilities.

But in case of remote operations, the ETICS system adopts the user's role and uses Grid technology to select the target machine, execute the tasks and get the results back. As mentioned in the first chapter, Grid technology has basically the same entities in all the different distributions, but the interfaces change so in order to use a major part of them, this layer also provides the way to send jobs to any supported Grid and to retrieve the results.

Resource tier

The resource tier is the lower level in the service architecture supporting the set of services of the business layer. It is responsible for providing data services to store data and metadata in a persistent way and access to computing services. In this layer, the technologies used in the current release of ETICS are grouped, but the previous design is precisely intended to make the system flexible to such changes without affecting the final service.

In terms of data services, this tier provides database service to store the project information and its corresponding metadata. Because of the nature of the data, it also provides more efficient storage capabilities for the metadata produced from the build and test tasks such as database systems or distributed filesystems.

In terms of computing resources, this layer encapsulates the platforms used during the build and test jobs either locally or remotely. In case of local scenario, the platform is the user machine so no additional action is needed. But in remote scenarios, the different requirements specified by the user make necessary to have the infrastructure able to analyze and select the target according to them (process known as matchmaking). In this case, the use of Grid technology appears naturally as the solution to manage the pool of available machines and the workload among all of them.

3.3 The ETICS facility

According to the description of the problem stated in previous chapters, the picture of the system given so far shows the ETICS proposal in terms of desired features and design principles. The data model, the service architecture or the test framework describe the information, mechanisms or good practices that are intended to transmit to the potential user and in a quite precise way. Hence, this is a very theoretical picture that needs to be validated building a real architecture, the proof of concept that the ETICS facility represents.

The ETICS facility is the actual infrastructure deployed by the ETICS project. It consists of a set of hardware and software resources that implement the data model and service architecture described above and that is shown in the figure 3.2.

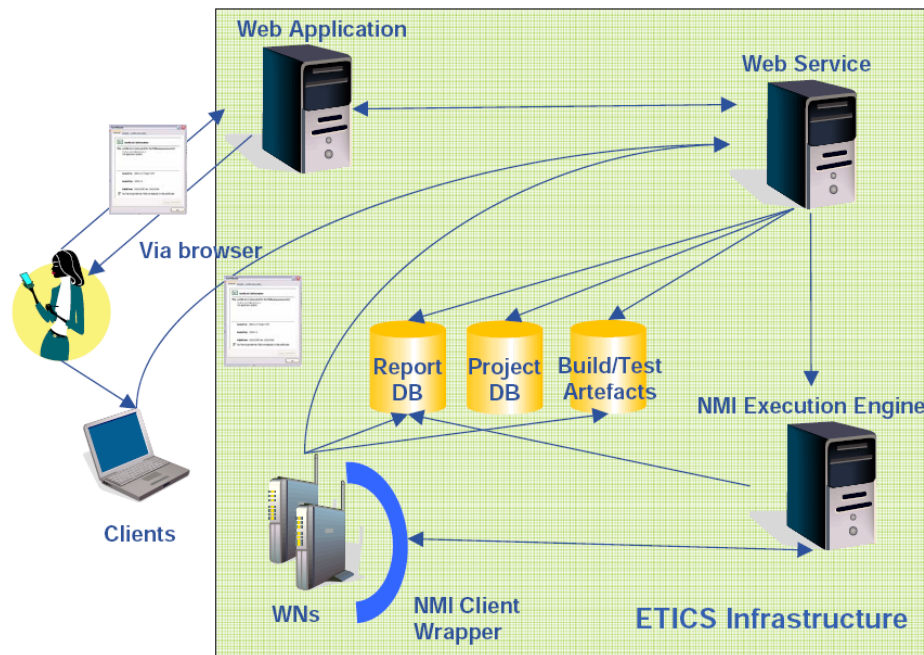


Figure 3.2: The ETICS facility

This figure depicts the entities involved in the service provision. It can be noticed that exactly matches the service architecture using a service distribution paradigm.

The core element of the system is the web service. This one performs the business logic and allows the user client to retrieve information to accomplish tasks. The web service is purely written in Java and is hosted within a Tomcat Application Container.

Towards the resource layer, the web service uses several abstraction layers in order to access such resources. Considering access to data sources, as already mentioned, the web service connects to data services. On one side, it stores directly the metadata gathered from the project definition into a RDBMS. This access must be RDBMS-independent, consequently, it is carried out using the Java ODBC library (JDBC). On the other side, the data and associated metadata coming from the build and test results is stored using Apache Jackrabbit as specific software for data libraries. Apache Jackrabbit is the fully conforming implementation of the Content Repository for Java Technology API (JCR) developed by the Apache Software Foundation.

Considering access to computing resources, in the remote scenario, the web service completely manage the access to these resources. The access to them is defined by mean of a generic interface to Grid Services. So the implementation of such interface will depend on the actual Grid infrastructure used. In the local scenario, the user client has access to the local infrastructure in user space using the native system libraries.

In terms of resources, the service architecture groups them in two main families: data and computing resources. Considering the first one, ETICS stores the information in two different services, depending on the nature of it. The metadata is stored in a Relational Database Management System, using MySQL Server for this purpose. This one allows to exploit the nature of the metadata as enables querying, chaining, etc. The use of an external service allows to control efficiently the data management and to get high performance. The data itself (artifacts) is stored in the distributed Andrew File System (AFS). This one has very good features by design like Kerberos security model, high availability (using data replication) or backup facility. The drawbacks are the variable level of performance, but despite of this, performed measures indicate that is still quite suitable considering its other advantages.

About computing resources, ETICS is designed to perform the final tasks (build and test) regardless if the target machine is the user's one or some remote platform in a Grid environment³. Therefore, the local scenario is the same as the remote one once the allocation has happened.

The current implementation of the ETICS system uses the tuple Condor/Metronome as the Grid enabler technology. This set of technologies defines a simple Grid environment where a central scheduler (Condor scheduler) receives jobs from submitter nodes (Metronome submitter node), locates the machine available that matches the some requirements (WN), sends the job

³What enforces one of the long term objectives: the reproducibility of results.

to it and retrieves the results giving them back to the submitter. In the WN, the Metronome job executes a sequence of stages where one of them is install the ETICS client to perform the build or test task.

Once the job lands on the target machine, the behavior is exactly the same as in a local machine. But the important difference is that the ETICS facility provides a fully managed set of platforms. The facility operates a set of more than 200 machines spread out of three of the consortium partners in the project (CERN, INFN, UoW), covering several architectures and operating systems. Specifically, the infrastructure managed by CERN is automatically managed using different software suites (automatic installation procedure with Quattor and AIMS, continuous monitoring with Lemon and SLS and daily backup of critical machines with IBM Tivoli Software).

Finally, the application layer consists of the three elements mentioned in the service architecture. In terms of CLI, both scenarios local and remote use the same user client. This one is implemented in Python and uses WSDL-generated libraries to access the WS (ZSI). It also allows to perform local operations to be committed later as well as data exportation in INI format.

In terms of web access, ETICS provides a web application to operate over project data, job submission, security administration or view information about results. This portal is implemented in Java and is also hosted in a Tomcat Application Container.

3.4 Summary

The ETICS project puts together different efforts and initiatives to setup a unified process for software development that ensures good levels of quality. As part of this effort, it aims to define the final structure that models any software development project in a modular and versatile way and to set up the mechanisms and channels to promote the broad dissemination of results for long-term analysis or storage.

Modules and configurations are the basic building blocks which allow to model the projects from the structural and dynamic point of view. The former ones provide an anatomical shape of the project itself, listing the software components and groups. The latter ones give physiological structure shaping the real structure, dependencies, relationships with platforms, procedural knowledge for build and test and keeping track of the whole history of the development.

This theoretical definition is implemented as a Service Oriented Architecture, taking advantage of the Grid paradigm to solve problems of resource dependency, allocation, security constraints and transparency in the service

provision. The four tier architecture (application tier, business tier, access tier and resource tier) decouples the metadata required in the project model from the physical resources required to implement it in the build or test phases.

The current ETICS infrastructure defines the set of services that perform the business logic for such task and that can be deployed upon any Grid distribution harnessing the already present technology to provide an added value service. Within this framework, an specific use case for distributed testing is designed as a proof of concept of the potential: the OGSA-BES.

Chapter 4

Test use case: OGSA-BES compliance

As seen in the section 2.1.2, software development projects define uses cases to fully describe the set of expectable behaviors of the system under development. Later, these use cases serve as a base for specifying the functional features of the software to be developed. The definition of use cases is also the starting point for testing activities because once defined, each scenario addresses a set of features that the system must provide and which must be validated according the user needs.

As a tool oriented to manage the full development process, the ETICS system provides an extensive set of features based on the definition of good practices (use cases) in the industry. As shown, these capabilities range from pure development process support (i.e. dependency resolution for building every component) to intensive testing (i.e. covering multiple testing targets and phases). Considering the testing functionality, a major feature provided by the ETICS system is the ability to reproduce actual scenarios where to perform production-alike tests. This one is a very complex task that can be splitted into five separated stages:

1. To describe the components (i.e. their required inputs and outputs) and their relationships
2. To define the automatic procedure which installs and configures the software components, leaving them ready to use
3. Following the initial timeline drawn by the dependency relationships, define and set up the mechanisms to coordinate the passing of configuration information between dependencies (messaging service)
4. To create the test cases and deploy them into the virtual environment

This chapter describes the distributed testing process using the ETICS system in a practical way. A priori, an interoperability testing scenario seems as the best kind of environment to demonstrate the powerfulness the ETICS systems aims to provide. This interoperability test case selected is also based on a Grid standard currently under development. Hence, the opportunity to exhibit the versatility of the system to build and test some part of its infrastructure itself confirms the choice.

In the next section, the selected proposal of standard is explained in more detail. Its motivation, its purpose and its technical aspects are also depicted to justify its election. Once this standard is presented, the specific objectives of the test case will be stated before proceeding to its development in the next chapter.

4.1 OGSA-BES description

In section 1.2, the technology fundamentals about Grid computing have been explained. It has been shown that computing services are one important component in any kind of Grid (i.e. services which provide access to computational resources). However, as it has also been drawn, every Grid has its own computing resource manager, with its own set of features, capabilities and access interfaces, showing that having different Grids, involves having different ways of sending jobs to each one. This diversity becomes a problem when Grid-enabled tools are developed because most of the times, they must be Grid-specific tools, allowing sharing of resources across the same virtual organizations but not allowing across different Grid schemas. This issue is clearly a problem with the own philosophy of Grid because the resource sharing is not really enabled for any kind of Grid-capable tool. Current circumventions are based on interoperation between components rather than the real interoperability between them that should happen by default.

In a trial to face up with this problem, the OGF has created a specific group to analyze the generic computing environment common to any Grid (OGSA-BES-WG). The goal of this group is to identify the common set of properties and operations performed in any Grid computing resource in order to define the common interface that guarantee minimum interoperability between users and resources, apart from specific functionalities provided by each implementation. The resulting common reference service is called Basic Execution Service (BES).

The BES is based on a common specification which defines the Web Services interfaces for creating, monitoring and controlling computational entities. Thus, this generic definition is not bound to any specific Grid

implementation, even to Grid technology. Clients define activities using the Job Submission Description Language¹[22]. The BES implementation will execute the job if it is acceptable in terms of requirements and usage policies, what depends on the underlying infrastructure. Consequently, the most important fact is that the BES web service is unaware of the specific implementation upon it works but characterizes it by publishing set of attributes.

To achieve the goals of creation, control and monitoring, BES defines three port-types, namely: BES-Factory, BES-Management and BES-Activity and a full data model, composed of state model, information model and resource model. Usage of all port-types will provide access to different implementations in a standard way while allowing exposure of platform-dependent differences possibly interesting to them. The implementation of the data model will ensure that data is managed in a consistent way and all the implementations will refer to the same entities and states.

4.1.1 Data model

The BES is part of the OGSA specification for modeling basic execution environments, that is, collections of resources in which a task can execute. The underlying environment, a queuing service, Unix host or any other specific resource is not shown to the user directly but through a set of generic operations and properties. The main objective is to provide the common framework to access such environments but keeping the implementations to expose their differences. For that, the BES model is designed to allow extensibility in the three main areas any computational environment covers: states, information and resources.

State model

IEEE defines state as *the set of values assumed at a given instant by the variables that define the characteristics of a system*. The state model represents the set of states, allowed transitions and trigger events of any dynamic system. The BES perspective perfectly fits into this definition since the standard proposes a detailed set of states, transitions and trigger events, and also specifies in which cases this model can be extended.

The main goal of this state model is to draw the schema followed in any execution task. The basic model fulfils this objective by defining the states shown in figure 4.1.

¹The language for describing the requirements of computational jobs for submission to resources developed by another working group inside OGF.

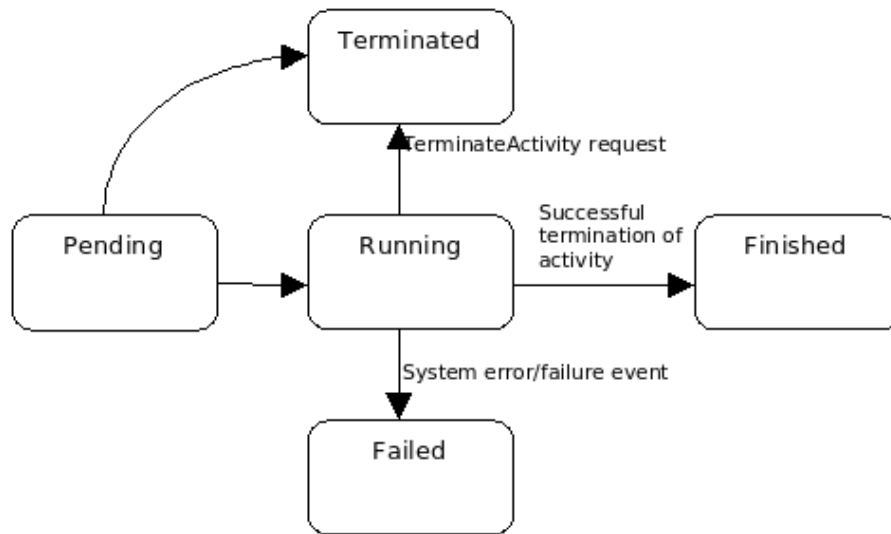


Figure 4.1: The PBSA-BES state model

- Initial states (pending): the service has created an entry in the system but not instantiated it on any computational resource yet.
- Intermediate states (running): the job is currently being executed in a suitable resource.
- Terminal states: the job has been terminated due to some of the following reasons:
 - finished: the activity is terminated successfully, where successfully means that was accomplished by the BES system not having any failure either in the BES system itself or in the underlying LRMS.
 - terminated: the activity is ended because of some external event such as user interaction.
 - failed: the activity is ended because of some error/failure event in any component of the BES system.

The transitions between this states are also fixed so:

- initial to intermediate: shows when the BES system initiate the processing of the activity and must always happen (even in cases of empty jobs).

- initial to terminal: shows the system aborts because of presence of errors or not supported operations in the incoming request to the BES system (not to the LRMS).
- intermediate to terminal: shows the completion of the jobs either successfully or unsuccessfully because of internal failures, bad requests or unmatched requirements.

Possible trigger events have been already pointed out. The most important is to notice that whatever the job is, all of them must pass through all the states (initial, intermediate and terminal) so the BES must detect the changes in the LRMS as well as the input events from the user interface to conduct the job and raise a failure in case of invalid request.

The BES specification also allows to implement custom extension mechanisms within the specific implementations. These extensions enrich the basic state model by adding internal sub-states that can mock up more accurately the behavior of the underlying resource. The inclusion of new sub-states is made via profiles².

However, the implementation of such profiles must satisfy certain conditions to ensure compatibility with standard user clients. Exactly three rules must be satisfied:

1. The sub-states defined in a new profile must not define transitions between states that are not already defined in the basic model.
2. The implementation of several profiles (composition) must not interfere with user clients which do not support any of the single profiles.
3. In case of composition of profiles, the elements defined in any of the profiles must be independent of the rest of elements defined in others. In other words, profiles must be concise in its scope.

Information model

The information model represents the data management required by the user to interact with a specific implementation of BES. It holds sets of different properties that map with LRMS properties, making available static and dynamic information like OS versions, types of executable files allowed, policies, QoS, etc.

²The Profile extension mechanism is a common way to extend a generic WSDL descriptor document in Web Services. Using profiles, specific implementations may extend the standard recommendation showing outside concrete additional details about the services they offer.

Its main objective is to serve as the abstraction layer that accommodates the differences mentioned in a single BES specification. Hence, it defines a set of attributes that any BES-capable implementation must support, establishing some of them as optional not to force very restrictive behaviors. It also allows to define additional attributes in a transparent manner for other non-compatible user clients.

The set of attributes defined by the basic implementation is part of the BES-Factory port-type are shown in the tables 4.14.2.

Resource model

The BES resource model represents the specific information that allows richer interaction with the LRMS. It is based on a profile mechanism to provide access to some concrete functionality. The only constraint precluded by the BES specification is that all BES implementations must support a simple operation for retrieving all attributes in a single document.

Some proposed extensions are related with idempotent execution semantics, subscription to notification events (e.g. WSRF, WS-Notification, WS-Eventing) or lifetime management (e.g. WS-ResourceLifetime).

4.1.2 Port-types

A port-type is a named set of operations and associated attributes offered by a Web Service. The BES specification defines three different ones, each one related with the sort of services provided: job processing, job monitoring and system management.

BES Factory

The BES Factory port-type sorts out operations that support the creation, monitoring and modification of collections of jobs. It also provides operations for retrieving attribute information from the BES itself. This last part might come into conflict with the BES Management port-type, but the original idea is to bind this set of operations to normal users and the actual BES management operations to administrator users since the former ones may need some information about the environment.

The attributes contained in this port-type must cover a wide range of resources or configurations so it adopts a flexible and extensible structure to describe the generics case. Hence, attributes are classified into two categories: basic resources and BES specific ones. A brief description about them can be found on table 4.1.

Name	Type	Multiplicity	Description
isAcceptingNew-Activities	BES	1	Availability status
CommonName	BES	[0..1]	Short human-readable name for BES
LongDescription	BES	[0..1]	Long human-readable description
TotalNumberOfActivities	BES	1	Current number of activities
ActivityReference	BES	[0..n]	EPR to currently managed activities
TotalNumberOfContainedResources	BES	1	Number of accessible resources
ContainedResource	BES	[0..n]	Document listing available resources
NamingProfile	BES	[1..n]	URIs of Naming profiles used by BES
BESExtension	BES	[0..n]	URIs of supported BES extensions
LocalResource-ManagerType	BES	1	Resource's LRMS type
ResourceName	BR	[0..1]	Resource's name
OperatingSystem	BR	[0..1]	Resource's OS type
CPUArchitecture	BR	[0..1]	Resource's architecture type
CPUCount	BR	[0..1]	Number of resource's CPU
CPU Speed	BR	[0..1]	Speed of resource's CPU (Hz)
PhysicalMemory	BR	[0..1]	Resource's physical memory size (B)
VirtualMemory	BR	[0..1]	Resource's virtual memory size (B)

Table 4.1: BES Factory attributes

Appart from the attributes, this port-type also has three built-in operations:

CreateActivity is used to request a new execution. The input of the activity consists of an XML document in JSDL format describing the activity and its requirements. On success, the output consists of the EPR reference for further reference to the activity. The method should raise an exception (i.e. SOAP fault message) in any of the following cases:

1. `NotAuthorizedFault` if the user does not have the proper rights to perform the operation.
2. `NotAcceptingNewActivities` if the BES is not accepting job requests.
3. `UnsupportedFeatureFault` if the JSDL document in the request is not well-formed or contains a non-supported element.
4. `InvalidRequestMessageFault` if the request specifies a non-recognized element.

GetActivityStatuses retrieves the current status for the activities already created and listed in the input EPR. The output contains an activity status document describing the status or an error message in case the request could not be accomplished. The method should raise an `UnknownActivityIdentifierFault` if any EPR provided cannot be mapped to any activity.

TerminateActivities requests a set of activities identified by their EPR to be terminated. The execution of this operation does not ensure that will be accomplished in such specific moment. The eventual success will be reported by consecutive `GetActivityStatuses` operations or by subscription and notification events. The output contains a set of responses, one per requested EPR, notifying the successfulness of the operation. If any EPR is not recognized or the operation cannot be executed, BES will raise an `UnknownActivityIdentifierFault`.

GetActivityDocuments retrieves the activity document sent during its creation. This document may be different from the initial one reflecting policies or internal processes executed. Therefore, it shows the activity being run rather than the activity asked to be run. The output contains the referred EPR and either the activity document corresponding to such EPR or the stack of faults raised during the retrieval. It may also contain an

Name	Multiplicity	Description
Status	1	Status of the activity
ActivityDocument	1	JSDL document containing the actual running activity
FactoryReference	1	EPR to the BES Factory resource responsible for the activity

Table 4.2: BES Activity attributes

UnknownActivityIdentifierFault if any EPR provided cannot be mapped to any activity.

GetAttributesDocument is used to retrieve the document with the list of BES Management attributes of this BES implementation.

BES Management

The BES Management port-type defines operations oriented to system administrators on their activity of managing the BES system. Currently, there are two operations to control the availability for incoming requests. Both perform their target operation without requiring input arguments and not giving back details about the success, which can be checked by the content of IsAcceptingNewActivities attribute.

StartAcceptingNewActivities to request that the BES starts accepting incoming request.

StopAcceptingNewActivities to request that the BES stops accepting incoming requests as soon as possible.

BES Activity

The BES Activity port-type is intended to define operations for monitor and manage individual activities. However, no operations are currently defined because most of the relevant information can be obtained from the BES Factory and no generic management infrastructure has been specified for OGSA services.

Nevertheless, there is a set of attributes that apply to individual activities. It is detailed in figure 4.2

4.2 Test objectives

As shown, the OGSA-BES recommendation establishes the general description of a Grid submission engine, defining the access to the common set of features that any of them implements and allowing additional mechanisms to extend the basic behavior. The use of a three-views data model is a clear example of the level of granularity intended in the performance and comprehension of the LRMS. In addition, it provides an objective description of the expected features that can be used as an interface with the systems during black-box tests.

Moreover, the OGSA-BES recommendation is a service description. This means that it does not come alone but requires the integration into a bigger environment with additional interactions with other parts or even other services. Hence, its testing will be performed firstly as part of the system tests for the services deployment of the Grid and secondly as conformance tests with the interoperability-oriented recommendation for the job submission Grid service.

Consequently, the previous description draws two general objectives closely related. Both focuses on different parts of the same certification process for the Grid development, that is, the testing of components according to standard recommendation and the fully automation of the process.

Interoperability testing based on standard compliance represents the conformance testing in the V-model for software development. In this case, the compliance testing refers to the ability of some component to provide the generic features stated in a general recommendation as a subset of its own functionalities.

The purpose of this kind of testing is to ensure native interoperability among different implementations of the same Grid concepts by checking the accuracy of such systems to the implementation of agreed common methods, shaped in the OGSA-BES. Therefore, the purpose is not to check the adequate definition of the recommendation itself in order to achieve interoperability but to trust the existing definition and evaluate the correctness of the implementations. This top objective requires to reach the following set of subobjectives:

- To develop a generic client used to perform all the service calls which the test is based on.
- To analyze the OGF recommendation for BES [23] in order to classify requirements and optional behaviors and develop the test.

The interpretation of the statements must be done according to the referred RFC [24].

- To select the programming language and framework that better suits to the case under consideration in terms of integration, performance and complexity.

Automatic deployment of Grid scenarios for testing covers the set of principles and required technologies to simulate production environments during the execution of tests over complete services. These kind of tests range from functional to stability or suitability tests. The final goal is to create the structure that, using the metadata already present in the ETICS system, is able to recreate automatically such scenario on demand, integrating the development of system tests of the developed services in the global QA process provided by the tool.

To address this general target, the next subobjectives need to be accomplished:

- To develop the pieces of software that will manage the deployment of the set of components involved in the service provision (formally called deployment modules).
- To design the structure based on metainformation that allows the execution of a test on a service by deploying its dependencies in an ordered way and that also allows the reutilization of common structures already present in other parts of the system.
- To ensure the quality of results by establishing the mechanisms to create the virtual environment where the tests can be executed achieving enough generality, with the required resources and software components and without compromising the execution (in terms of security and shared access to resources) of other tests running simultaneously.
- To harvest all the information produced during the tests and present it in a unified report form which can be stored in a repository for further analysis or comparison with latter results.

Although these objectives are clearly disjointed by themselves, the fulfillment of both provide the framework that integrates seamlessly an important part of the QA process stated in previous chapters. Upon completion of this proof of concept, the latest release of some Grid service shall be tested automatically against the public recommendation for such service, and hence,

two important benefits will arise: firstly, on successfully tests, the interoperability among compliant services will be qualified with empirical results and secondly, the whole QA process will be improved by accomplishing well-defined guidelines after using the ETICS facility out-of-the-box.

4.3 Summary

The ETICS system features the ability to deploy automatically virtual scenarios for fully controlled distributed tests. This power of this capability aims to be demonstrated with the implementation and deployment of the compliance testing of the OGSA recommendation for job submission services called *Basic Execution Service*.

This recommendation defines the basic functionality that must be offered by any compliant service in order to normalize the interface that connect to the underlying LRMS. For that, the recommendation sets a three-view data model comprised of state model (i.e. states definition and transitions), information model (i.e. generic information to access any resource) and resource model (i.e. extension to the generic functionalities to use specific options of the underlying resources).

According to the data model, the port-types are defined (bound to the kind of services offered). The BES-Factory port-type interfaces to control the creation, monitoring and modification of sets of activities. The BES-Management port-type operates the local LRMS and is intended for administration purposes. Finally, the BES-Activity port-type defines the attributes that describe the state of the engine and activities being performed.

This description shows a service interface whose implementations will be supposed to be interoperable by default. Therefore, the main purpose of the integration into ETICS is to check interoperability between services verifying their accuracy to the common definition, the OGSA-BES. But also, as part of the integration into the ETICS system, the testing process will be fully automated using the current facility for virtual testbed deployment. Success on such task will improve the QA process by testing implementations of services under development exactly against their final specification documents.

Chapter 5

Test use case: design, implementation and results

Starting from the Grid as the new paradigm for distributed computing and efficient sharing of resources, previous chapters have depicted the current difficulties found in its development and deployment. The appearance of different communities around the world focusing on several aspects of this theoretical model has produced great results but they have not always been able to work together.

Also in previous chapters, the software engineering techniques have been shown, describing their advantages in order to produce the predictable and high-end results that can be observed in other science disciplines. Nevertheless, this is not an easy task and currently there are many efforts in order to automate these processes across any software development project, and therefore the Grid development as well. Considering the latter one, several initiatives have targeted the objective of defining and enforcing the guidelines that ensure minimum quality levels in final releases.

Regarding this environment, this chapter describes the concrete set of activities accomplished to reach the objectives stated in the previous section, namely, to ensure interoperability among services by performing automated tests for standards compliance. Across the next sections, design considerations, implementation details and final integration will be described. In addition, last part will cover results and preliminary conclusions, prior to the final dissertation made in the last chapter.

5.1 Test design

The test design phase must cover every aspect of the test, from what to test and how to where to perform and how to schedule every action. Each one of these questions is answered by different entities which represent different tasks of the test. The test environment presented consists of two independent actors that perform a well-known set of activities. These ones are classified according to the purpose and scope of their contribution to the final result. These two entities exactly matches the two main objectives described previously: on one side, the deployment of the services to be tested automatically (i.e. the testbed) and on the other side, the deployment of the test against such selected services (i.e. the client).

The ETICS system will be responsible of launching and coordinating the execution of both entities by providing the mechanisms that enable automatic deployment and communication among activities (messaging services). The structure provided by ETICS allows to specify by mean of configurations and dependencies the whole test structure in order to select dynamically (runtime) the proper service to be tested on each moment. This behavior also involves some assumptions in the way the code is structured allowing better reuse of parts and modularity.

5.1.1 The client

The client is the entity of the test whose objective is to connect to the service under test (IUT) and make the calls according to the predefined schema. Hence, it is responsible of what to test and how to do it.

From a theoretical point of view, in order to use any service and perform remote calls four elements are required to be known: the description of the functions implemented (i.e. the interface), the way these ones can be accessed (i.e. the set of messages allowed), the description of the expectable states or workflow of operations (to keep track of the order and sequence of tasks) and the policies of usage (e.g. authentication and authorization).

With the upcoming of the WS paradigm, the two first elements can be addressed by the usage of the WSDL. This one simplifies the process of discovering all the functions, their arguments and results and the format of the messages to be exchanged. Depending on the programming language selected, there are different alternatives to deal in an automatic way with these documents and the generation of the exchanged messages. In such cases, the WSDL document can act as a source definition document in order to generate the language-specific code (i.e. libraries) that implements the interface declared in the document from both the client and server sides.

As can be guessed, this option greatly facilitates the dealing with WS in a programmatically way, simplifying the remote calls like local ones (the underlying framework will translate those calls to the proper messages invoking the corresponding remote functions).

In the case of OGSA-BES service, the recommendation [23] defines a normative WSDL document for each port-type. Because of being normative, its usage must be compliant with any OGSA-BES implementation and therefore, it can be used as the first requirement for a successful test. So, instead of discovering the functions offered by the target service, the client will use its own version of the WSDL, the normative one, to make the calls.

The third characteristic element of a service is the allowed set of states and triggering events. By itself, it helps to perform new activities coming from the same client with better allocation of resources and simplifying the call. Some of the key points during the definition of some protocol are the establishment of the states, the transitions and the synchronization between entities talking the protocol. In addition, WS technology is not session-oriented by default. Consequently, several mechanisms have been pointed out to address this problem, like WS-Session or the session management from the server side. In terms of clients, this issue usually requires no more effort than keeping track of some session identifier provided by the server. However, the management of such information and its life cycle plays an important role in the whole process.

The OGSA-BES recommendation defines a complete data model which precisely aims to cover all these aspects of the job submission. This state model will be one of the targets to be tested by the client, checking the proper definitions and transitions between states. Nonetheless, as it is based on WS there are no sessions formally. The only reference to previous calls managed by the client is returned as an EPR¹. Therefore, the client will have to keep such information and manage it properly during the execution of the test.

Finally, and closely related with the state management, there are the authentication and authorization mechanisms. The absence of state in WS by default has the consequences of either requiring authentication on every query or develop the proper tools and processes for security management.

The OGSA-BES does not cover any specific security policy schema more than the required messages to be exchanged in case of security violations. Therefore, the execution of remote calls within BES-capable endpoints will depend on the implementation. Again, this decision involves that the client

¹An EPR is a unique identifier that points univocally to an activity. It is defined by WS-Addressing and its format depends on the specific implementation but generally consists of the URL of the service and the internal identifier or session id.

must be able to deal with different security schemas and services in order to be able to perform the test. Specifically inside Grid environment, it must be ready to interact with the common schemas such as SSL, WS-S, proxy certificates or against VOMS or Kerberos services.

Tests

The client previously described embeds the required logic to connect and perform operations against any OGSA-BES WS. Although the ability to connect using the normative WSDL is part of the test, the test is formally performed by a compliance testsuite.

As explained in section 2.1.2, a compliance testsuite is a composition of simpler tests that check the compatibility of some implementation against the standard document. This recommendation is usually written using a combination of natural and formal languages. The natural language is ordinary language used to describe the behavior and models in a conceptual way. The formal language uses mathematical or machine readable notations and is used to describe explicitly the actions, processes and expected results. In the case of the OGSA-BES recommendation this mixture effectively happens. Natural language is used to describe the state model and formal language is used for the information and resource models (pseudocode and normative XML-format documents).

In order to code this specification into executable code, two considerations must be taken:

- use of the RFC2119 [24] to interpret univocally the meaning of common keywords (must, should, may, etc.)
- use of programming language and tools to translate to machine processable language the formal specifications

Keeping this in mind, and considering the service nature of the IUT, the more reasonable way to perform the compliance test is by doing functional testing against the selected endpoint. Testing the functionality of some service according to its specification means to test boundaries, unsupported options, privileged behaviors, incomplete queries and finally normal or expectable requests.

Because every test checks one concrete aspect of the functionality of the service, each one must be self-contained. The more uncorrelated with other tests the clearer and more accurate the results will be. This concept of testing is very close to the basic idea under unit testing: performing some operation and asserting the results. Hence, the usage of unit testing frameworks

appears as the best option to create a simple testsuite as a collection of unit tests, but understood as functional tests and to present the results in a uniformed way.

Once the client and the test are designed, they will be integrated in the testing structure that launches the service, execute the test and retrieves the results: the testbed.

5.1.2 The testbed

The testbed is the other entity of the test. Its objective is to setup completely the scenario where to run the IUT, the service, and the test client to generate the compliance reports.

In order to achieve this goal (already specified in previous sections 4.2), the system must be able to:

- select and reserve the resources that match the service requirements
- analyze and resolve dependencies between service components in terms of packages and metainformation for runtime operations
- deploy the service and launch the test and scratch it once the test is finished (whatever the final status is)
- harvest the results and archive them

Currently, the ETICS facility is able to select target machines that satisfy some requirement. Some of this requirements can be specific platform (architecture and OS), specific configuration (presence of concrete software components), root access, private use, etc. The ability to provide this feature is usually bound to scheduling capabilities because in case of finite number of resources, the queuing mechanism is a good way to get results by introducing some delay. While this is useful for build activities, for distributed testing is not the best approach since the reservation of some resources and the wait of others for being attended is unstable and may cause resource starvation. Therefore, the system must include another working mode for tests, the central scheduler must provide bundle of resources on demand for the test requirements. If there are not enough resources, the whole query must be either hold on until the availability of the busy ones or completely discarded.

The dependency resolution feature is already present in ETICS. With it, the system can analyze the full tree of dependencies and associated information with them such as version numbers or installation paths. In the case of test jobs, the system must be able to provide the packages to install the

service on some target machine and deploy the specific repository technology usable by the OS (APT, YUM, etc.). Therefore, something to add is such automatic deployment containing the packages resolved from the dependency resolution.

The service deployment is the key point of the testbed. Once the system knows the packages that will be used in the test, it must launch the installation of the participating services but following the order established by the dependencies. In other words, following the dependency resolution, the ETICS system must launch a process on separate machines for those services that cannot run on the same one and act as a messaging gateway to communicate the configuration information.

As ETICS aims to enhance the modularity and reusability of software, in case of testing this desire requires the definition of the suitable structure. For this, the following set of new modules and configuration entities has been defined:

deployments are the description of a top-level service like a web application or a computing element for the Grid. They consist of nodes or other deployments.

nodes are modules that contains simpler services that make part of a bigger one like the database for a web application or the scheduler in a Grid CE environment. Nodes also represent entities that perform tests against services (in a deployment or node contexts).

services are modules that represent a concrete instance of a single service. These ones contain the information and procedures to deploy the underlying service on a target machine. They exactly match the concept of ETICS configuration in a build context.

tests correspond to the analogous entity as the service module but regarding the testing scope. While the service is the IUT and runs in a daemon manner, the test represents the client using the service and generator of results.

Accordingly to this structure, the implementation of the OGSA-BES compliance test must be automated by developing the suitable deployment modules for the service under consideration and by modeling the deployment in the ETICS system.

Harvesting of results will be done by the ETICS client and sent back to the central repository as in the build context. The difference is that such report must merge the results of the different test modules and include log information from the rest of modules.

Deployment modules

In the case of the OGSA-BES compliance testing, the deployment of four nodes is required. Three of them contain different parts elements of the global service and the other one execute the test itself. Next, those services (grouped by their scope) and their deployment are described:

repository contains the specific services that allow to act as a repository for the selected package management system (as mentioned, APT, YUM, etc.). Its deployment requires to access the ETICS system, download the set of packages used during the test, create the folder structure and metadata understandable by the package management system and finally launch a web server that provides access to all this data.

worker node represents a computing power resource, the place where jobs will be run. In order to deploy it, access to the previously launched repository is needed, and later on, the installation and configuration of the daemon which will be connected from the dispatcher to execute jobs.

computing element represents the entry point to the computing power resources, queuing and dispatching jobs over the pool of attached worker nodes. Depending on the Grid implementation, it will require access to different services, but basic services are information and AuthN/AuthZ services. Because of registration of resources, the information services must be deployed within the testbed, but others are out of the scope of the test and can be used from external sources.

The forth node will contain the compliance test. It must wait for the rest of services (the CE endpoint) to be ready before launching the test. Once done and the test executed, this node is responsible for launching the removal process to retrieve the results and remove the services.

When a software component requires some other for its normal working, it is possible to define dependency relationships on it. But when a service depends on several software components to provide its functionality, although any of the components are able to work independently, the way of link such packages is through the metapackage mechanism. The ETICS system models these metapackages as an empty package whose dependencies are the different components required for the global service provision.

From this design, several issues arise as a matter of security risks and race conditions. These ones depends on the implementation and the technologies used and can be relieved using the different techniques detailed in the following section.

5.2 Test implementation

Relying on the design phase, the implementation of the test uses its design principles in order to create software structure and components that finally allows the execution of the OGSA-BES compliance test.

According to the structure designed, the implementation will consists of three stages: the development of the client and the test code, the development of the deployment modules to launch the services under test and the adaptation of the ETICS system to integrate these new features of automated distributed testing.

5.2.1 Compliance test code

As stated in the previous section, the test consists of two components, the generic client and the test itself, performed by such client. Both must be easy to deploy and integrate since minimizing the complexity will maximize the performance of the test.

From the client side, the use of WS technology is intended to simplify and enhance the communication between clients and services. As seen, the service is defined by its interface, described through a WSDL document. This document is taken as the contract between the two parts, client and server so the internal code of both (i.e. signature of methods and types of arguments used) can be automatically generated with different tools. Despite WS are designed to be loosely coupled to the transport technology, common practices includes usage of the SOAP protocol over HTTP channels to exchange messages between parts. Therefore, the utilization of the proper tools and frameworks simplifies even more this communication.

This is the case of Apache Axis. It is an implementation of the SOAP protocol that offers a complete framework to develop WS from both sides, abstracting concrete details of the protocol implementation. It also offers a code generator to, starting from the WSDL document, generate the *skeleton* of classes that perform the functionality in the server side as a library of functions that, used in the client side, provide access to the service functionality like a local library (formally called *stub*). It is purely written in Java so using it, will involve to use Java as a programming language for the client. Consequently, the first component of the client is the stub generated with Axis.

Although, the ETICS integration will be explained later, to become integrated, any component must be completely operational from the CLI. Regarding Java applications, Apache Ant is the common utility (like GNU Make) that allows to create scripts that automate frequent processes of

some component by the definition of variables and targets. The Axis code generator includes an Ant extension to generate code from a WSDL document as a target. So merging both tools, the stub can be completely generated on demand.

Once the remote functionality of the WS is locally accessible, the next step is to develop the collection of utilities surrounding it and which provide added value. Such utilities are possibility of different authentication schemas (HTTPS, WS-S and GSI), parametrization of the behavior using configuration files, logging facility and wrapping the WS methods putting all together. With this, the generic client is ready to connect to the available set of OGSA-BES services and the last step is to develop the compliance test.

The test is designed to test boundaries, unexpected input values, etc. of the two available port-types of the OGSA-BES service: BES-Factory and BES-Management². For each port type several test are defined to test every method. The table A.1 in appendix A.2 lists all the tests of the testsuite and gives a brief description of each one.

As mentioned in the design, all these tests are intended to verify the minimum functionality that any OGSA-BES implementation must provide. Nevertheless, they are implemented as a suite of unit tests using the unit testing framework for Java, JUnit. This one is a simple tool easy to deploy, fast to develop and whose integration in other technologies like Ant or ETICS is available through extensions. The unit tests are grouped in a suite which executes all of them sequentially. The final report is a javadoc-style document parsed to HTML using the JUnit Report module. Like the generation of the stub, this test execution can be launched from the CLI using Ant, what enhances the integration within ETICS.

Once the test is in place, the next step before executing it is to develop the deployment modules and utilities to automate the process of deploying it on demand an a clean dedicated set of machines.

5.2.2 Deployment of the scenario

The final goal of the test is not only to validate the implementation of any OGSA-BES service under development but also doing it automatically. Automate the deployment of any service faces up with two major issues: automatic installation and automatic configuration of the software.

Automatic installation of software means to know exactly which components are needed and where they must be installed. Currently, use of package

²The BES-Activity provides attributes included in the other two port-types but does not offer an endpoint to which connect

management systems solve this issue because the package contains all the information. However the trouble that arise is the security requirements imposed to perform such task. In most of the systems, system folders are protected and although the install directory could be changed, the services still need privileged access to some resources. The usage of virtualization techniques appears as a natural solution to address this issue.

Currently, several software products provide virtualization utilizing different techniques (emulation, hardware virtualization, paravirtualization, etc.). The final features are similar, distinguishing on the internal behavior. Choosing one or another can impact on the global performance of the test or on the complexity of the deployment. The lightness, easiness, wide support and availability of the open source alternative, Xen, and the portal vGrid for its management, make it quite appropriate for this case.

The second issue, configure automatically software components, requires to know the interactions between components and their dependencies, what means the information needed by ones must be available to use the functionality provided by others. The procedure of configuring some component can be fixed, not the information used. This parametrization gives simplicity and provides modularity to use the same configurator with different components just changing some parameter.

The OGSA-BES service consists of three simpler services that are deployed using their corresponding deployment modules. Any of them control the life cycle of the component, therefore they may receive startup and shutdown values as input arguments. Next, a more detailed explanation is given for each one:

repository is mounted using a Python program to fetch the metapackages and the lists of packages required during the installation. ETICS can generate empty packages whose purpose is to link to others in order to define the set of programs that make a service. Therefore, installing one metapackage, all the related packages are installed.

This module runs the repository generator, RepoJanitor, that creates an APT/YUM repository (suitable for the Linux distributions used in the test). It also installs an Apache Web Server to be run on the specified directory (i.e. in the user home directory not the default path) and launches it. When the service is up and running, it publishes the location of the idle repository in URL form using the ETICS messaging service.

Finally, upon completion of the test, the module stops the server and delete the repository.

worker node is launched using a Bash script which installs the software and configures it to be attached to the dispatcher node where the OGSA-BES application is targeted to run. For installing the software, the module reads from the ETICS messaging service the location of the repository. It also needs to read the target machine where the dispatcher is being installed. The configuration in the dispatcher requires to know where are the worker nodes so it also publishes its location.

As in the case of the repository, the service is stopped when the test is finished and the used space is scratched.

computing element groups the LRMS and the OGSA-BES service under test. It is installed once the repository is available (getting the information from ETICS) and configured when the worker node is ready. Upon completion, it publishes the URL to the test where the WS is accessible.

The reception of the removal signal is acknowledged stopping the service and releasing the resources.

compliance test is not a service itself so there is no installation stage. The configuration required is minimal and only requires to know the OGSA-BES endpoint enabled to perform the compliance test.

Depending on the Grid distribution, the client may require to publish some information before accomplishing the task because some operations need to be performed on the service side³. In such case, the deploy of the service waits for the client and its information to be published.

When the test finishes, the client saves the results and ends. The ETICS client notices the exit and sends the results back to the central repository. After, it sends the removal signal. The rest of nodes participating the test catch it and start the removal process mentioned before.

Prior to execute the test, the ETICS system deploys adapted Xen virtual images for every required node via vGrid commands. The virtual machines are completely manageable from CLI so their life cycle centrally managed. In case of failure (i.e. on the test or on the synchronization), the timeout running on the head node runs out and the physical machine is ordered to stop the virtual one. This behavior runs as part of the timeout mechanisms

³To use some privileged operations, gLite CREAM-BES requires the client DN during the configuration stage.

for the tests implemented by ETICS and is completely transparent to the user.

The manner of running jobs on these nodes follows the push model. Once the nodes are loaded, in final step of the boot process consists on communicate with the central service in order to get the description of the activity to perform. The information passed to the machine contains the messaging session identifier, the project and modules to checkout and test and the remote place where copying back the output upon receiving the termination signal. With that information, the habitual ETICS process is launched, that is: setup of the workspace and project, checkout of the code required for the test and execution of it (appendix A.1).

Within it, deployment scripts are executed for each service and they get synchronized sending information to the others using ETICS commands (listing A.3).

In order to recreate this scenario and use these sources, the ETICS system must be set up with the metainformation that describes the procedures to execute. Such information models the behavior of the project and is detailed in the next section.

5.2.3 Modelization in ETICS

The utilization of the ETICS facility in the performance of the OGSA-BES compliance testbed covers two aspects of the work: the build and packaging of the client and the test (using the build facility) and the execution of it (using the test facility). The information required in both cases for the description of each activity is stored in the project *Grid testbed compliance*. It contains the subsystems and components that model the activities to do and, in the current version of the system, contains configurations as the smallest information bricks.

The testing client is represented by *eu.omii.bes* subsystem which consists of four ETICS components: *stub*, *utils*, *clients* and *compliance*. A more detailed description is presented in the next paragraph.

eu.omii.bes.stub generates the stub which connects to every BES endpoint. It builds for any platform the *bes-stub* package and has dynamic build dependencies on the *axis* and *ant* packages.

eu.omii.bes.utils contains additional utilities such as security handlers or parsers to perform the tests. It generates the *bes-utils* package and depends on the following components: *axis*, *ant*, *bouncycastle*, *log4j*, *wss4j*, *xml-security*, *glite-security-trustmanager* (GSI), *glite-security-util-java* and *eu.omii.bes.stub*.

eu.omii.bes.clients represents the generic OGSA-BES client based on the normative specification ([23]). It builds the `bes-generic-clients` package and depends on `axis`, `log4j`, `ant`, `eu.omii.bes.util` and `eu.omii.bes.stub`.

eu.omii.bes.compliance contains the suite of compliance tests for the OGSA-BES service. It also contains the executable part of the tests and depends on the previous packages. It also depends on `axis`, `log4j`, `ant`, `junit`, `jdk` and `eu.omii.bes.clients`. The package it builds is `bes-compliance-tests`.

Apart from this description, any of the components mentioned contains the same commands to indicate the system how to checkout the code (i.e. VCS Commands) and the instructions to build the package (i.e. Build Commands).

For the deployment of the testbed, the subsystem *eu.omii.compliance* groups the components and configurations that correspond to the deployments, nodes, services and test described in the section 5.1.2. Despite such structure is not available in ETICS as described, because of being developed parallel to the develop of this test, the current structure is functionally equal and behaves in exactly the same intended way as the new one. Hence, the subsystem `eu.omii.compliance` acts as the deployment; the components *eu.omii.compliance.repo*, *eu.omii.compliance.wn*, *eu.omii.compliance.ce* and *eu.omii.compliance.client* are the nodes; and their configuration dependencies are the services and tests that participate on the distributed test.

This structure is perfectly suitable for the requirement of being able to test different OGSA-BES services. By creating several configurations for each component, one per service, and linking all of them in a subsystem configuration, the system replicates the test structure for the target service.

For instance, for the OGSA-BES implementation of gLite (CREAM-BES), the *eu.omii.compliance.glite* configuration is created. This one has four subconfigurations, one per component, each one of them depends on each one of the components of the deployment modules for the gLite services (the subsystem *eu.omii.compliance.glite*):

eu.omii.compliance.repo.glite holds the instructions and parameters to deploy a repository service that delivers the packages required in a gLite installation. It lists the metapackages to be downloaded from ETICS when running the repository and depends on the *apt-deployment-cream* component from the *Externals* ETICS project to create the repository.

eu.omii.compliance.wn.glite includes instructions about how to checkout the deployment module for the gLite CREAM CE worker node. It does

not depend on any external package because its deployment module just needs to download the *pbs-mom* metapackage from the ad-hoc repository and configure it.

eu.omii.compliance.ce.glite contains checkout instructions to retrieve the deployment module for the gLite CREAM BES service. Like in the case of the worker node, the deployment module is self-contained and does not need any external dependency.

eu.omii.compliance.client.glite comprises checkout instructions to deploy the compliance test on the gLite CREAM-BES service (essentially the same for any other OGSA-BES service), so there is no external dependencies.

During this explanation, use of metapackages has been mentioned as a way of grouping closely-related packages. The used ones for the gLite CREAM-BES compliance test are:

- *eu.omii.metapackages.testsuite* groups the test client packages
- *org.glite.metapackages.CREAM-BES* groups the components of the CREAM service (CREAM, CREAM-BES, BDII, TORQUE-server, etc.)
- *org.glite.metapackages.TORQUE-server* groups the components for the PBS scheduler service
- *org.glite.metapackages.TORQUE-client* groups the packages for the PBS scheduler service client

Consequently, regarding the model presented for the distributed testing within ETICS, a configuration is used with two different meanings: on one side, for the build part, as a version control mechanism of the resulting package of the component (like the usual version mechanism in any VCS source) and on the other side, for the testing part, as a branch from the central concept of OGSA-BES compliance testing, focusing on the specific distributions under test (like in the branch concept of VCS sources).

This one has been the explanation of the implementation of the compliance test, from the points of view of the test development and the testbed compliance. Next, the results of its execution for the gLite CREAM-BES service are detailed.

5.3 Test deployment and results

After designing and implementing the compliance test and the testbed with ETICS the next and final step is to execute it and get the results.

The OGSA-BES compliance testing can be executed from the ETICS WA or from the ETICS CLI. The concrete commands used for its execution are shown in the appendix A.1⁴. Under such scenario, table 5.1 presents a short list of the results of the test.

Although these results are specific for a single Grid distribution, they are a good indicator of the surrounding circumstances of a standard implementation in a Grid environment. The alpha status of these releases suggests that the implementation is still under development and also in this case, it implies that the recommendation itself is under public comment phase.

As it can be seen in the figure 5.1 and appendix B.2, the test has been successfully deployed. The system has managed to schedule several jobs, deploy the corresponding virtual machines for each of them, launch the individual jobs, act as a messaging gateway communicating them and retrieve the results archiving them in the ETICS repository.

Therefore, despite of the errors in the test performance, the final results is that the compliance test itself has been successfully accomplished. Using the ETICS facility, it has been able to execute compliance tests against latest releases of several services under development (gLite CREAM and CREAM-BES) and also setup the scenario for the test. Nevertheless, the specific consequences of these unsuccessful compliance test results as well as issues during the execution and possible future works are covered in the next chapter across the conclusions of the work.

⁴The reader shall notice that because of availability of the releases, the test has been executed only with the gLite CREAM-BES service.

Designed for use with [JUnit](#) and [Ant](#)

Unit Test Results

Class eu.omii.bes.compliance.BESComplianceTestSuite						
Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
BESComplianceTestSuite	22	22	0	77.162	2007-09-24T16:04:51	kb1303v1.cern.ch

Tests						
Name	Status	Type	Time(s)	Time(s)	Time(s)	Time(s)
BadFormattedJSDLAactivityDocument	Error	Unexpected exception, expected<org.omg.bes.factory.UnsupportedFeatureFaultType> but was<eu.omii.bes.exception.ServiceInvocationException> java.lang.Exception: Unexpected exception, expected but was Caused by: eu.omii.bes.exception.ServiceInvocationException: nested exception is: java.lang.IllegalArgumentException: Invalid table element 'jobdefinition' is null at eu.omii.bes.clients.FactoryClient.Creativity(Unknown Source) at eu.omii.bes.compliance.factory.BadFormattedJSDLAactivityDocument(Unknown Source)	0.506			0.506
UnsupportedJSDLAactivityDocument	Error	Unexpected exception, expected<org.omg.bes.factory.UnsupportedFeatureFaultType> but was<java.lang.NullPointerException> java.lang.Exception: Unexpected exception, expected but was Caused by: java.lang.NullPointerException at org.omg.bes.clients.FactoryClient.Creativity(Unknown Source) at eu.omii.bes.compliance.factory.CreativityTest.UnsupportedJSDLAactivityDocument(Unknown Source)	0.107			0.107
UnsupportedFeatureActivityDocument	Error	Unexpected exception, expected<org.omg.bes.factory.UnsupportedFeatureFaultType> but was<java.lang.NullPointerException> java.lang.Exception: Unexpected exception, expected but was Caused by: java.lang.NullPointerException at org.omg.bes.clients.FactoryClient.Creativity(Unknown Source) at eu.omii.bes.compliance.factory.CreativityTest.UnsupportedJSDLAactivityDocument(Unknown Source)	0.112			0.112
PrivilegedActivity	Error	Unexpected exception, expected<org.omg.bes.factory.NotAuthorizedFaultType> but was<eu.omii.bes.exception.ServiceInvocationException> java.lang.Exception: Unexpected exception, expected but was Caused by: org.omg.bes.util.FormatConverter.getMessageElement(Unknown Source) at eu.omii.bes.compliance.factory.CreativityTest.UnsupportedFeatureActivityDocument(Unknown Source) at eu.omii.bes.compliance.factory.CreativityTest.UnsupportedFeatureActivityDocument(Unknown Source)	48.333			48.333
StoppedService	Error	Unexpected exception, expected<org.omg.bes.factory.NotAcceptingNewActivitiesFaultType> but was<eu.omii.bes.exception.ServiceInvocationException> java.lang.Exception: Unexpected exception, expected but was Caused by: eu.omii.bes.exception.ServiceInvocationException at eu.omii.bes.clients.FactoryClient.Creativity(Unknown Source) at eu.omii.bes.compliance.factory.CreativityTest.PrivilegedActivity(Unknown Source)	0.855			0.855
InvalidRequest	Error	Unexpected exception, expected<org.omg.bes.factory.InvalidRequestMessageFaultType> but was<eu.omii.bes.exception.ServiceInvocationException> java.lang.Exception: Unexpected exception, expected but was Caused by: eu.omii.bes.exception.ServiceInvocationException at eu.omii.bes.clients.FactoryClient.Creativity(Unknown Source) at eu.omii.bes.compliance.factory.CreativityTest.PrivilegedActivity(Unknown Source)	0.605			0.605

Figure 5.1: Compliance tests results page

Table 5.1: Test methods results for gLite CREAM-BES

Name	Status	Description
BadFormattedJSDLActivityDocument	Error	Unexpected exception, expected <org.ogf.bes.factory.-UnsupportedFeatureFaultType> but was <eu.omii.bes.-exception.ServiceInvocationException> Error produced because the Axis implementation of the SOAP messages does not allow null messages (inherited from the WSDL file).
UnsupportedJSDLActivityDocument	Error	Unexpected exception, expected <org.ogf.bes.factory.-UnsupportedFeatureFaultType> but was <java.lang.NullPointerException> Error produced because of usage of the JSDL Posix profile which is not supported in the WSDL document of the test client.
UnsupportedFeatureActivityDocument	Error	Unexpected exception, expected <org.ogf.bes.factory.-UnsupportedFeatureFaultType> but was <java.lang.NullPointerException> Error when requesting an activity whose requirements are properly formatted but no sense with the request.
PrivilegedActivity	Error	Unexpected exception, expected <org.ogf.bes.factory.-NotAuthorizedFaultType> but was <eu.omii.bes.exception.-ServiceInvocationException> Error trying to perform an activity which requires special privileges to be run on any OS.

Continued on next page...

Table 5.1 – Continued

StoppedService	Error	<p>Unexpected exception, expected <code><org.ogf.bes.factory.NotAcceptingNewActivitiesFaultType></code> but was <code><eu.omii.bes.exception.ServiceInvocationException></code> Error stopping the service and checking if new jobs are accepted (the stop call is not implemented).</p>
InvalidRequest	Error	<p>Unexpected exception, expected <code><org.ogf.bes.factory.InvalidRequestMessageFaultType></code> but was <code><eu.omii.bes.exception.ServiceInvocationException></code> Error submitting a job that launches an inexistent program.</p>
NormalActivity	Error	<p><code>eu.omii.bes.exception.ServiceInvocationException; nested exception is: org.xml.sax.SAXParseException: Premature end of file.</code> Internal server error when performing a normal operation. It is probably due to a configuration issue.</p>

5.4 Summary

The implementation of the OGSA-BES test use case presented in previous chapters has as main objectives to perform interoperability tests verifying the compliance to the recommendation automatically. All the process is lead inside ETICS but two different activities happen for that.

Firstly, the OGSA-BES compliance tests is designed according to the statements of the recommendation and the WSDL descriptor document. Therefore, this test verifies as a suite of unit/functional tests, that the methods and attributes are accessible as specified and returning results as specified for certain conditions. As well as testing the exceptional behaviors, the test runs the usual requests.

Secondly, the test must take place in a fully operational service. This is accomplished by setting up the target service on demand, previously to the test execution and in a completely automated way. For that, the ETICS system is used, modelling the components that build the service and preparing the clean machines on which they are installed.

Exactly, the test is executed over the gLite implementation of OGSA-BES, the CREAM-BES service. For that, the testbed is execute with four nodes consisting of a repository, a worker node, the computing element and the test client. The ETICS system is properly configured to identify the services to be run on each node. The test is successfully executed getting the results shown in the table 5.1.

Chapter 6

Conclusions

As seen across the first chapters, software development is not so easy task. Part of the reason is the big complexity of settling and defining all the possible states allows to happen under some circumstances, but another important part is the difficulty to code that in a formal language and check the correctness of such implementation. In addition, in the case of the Grid paradigm, software development inherits the typical issues but it also adds the great troubles of designing the stack of protocols and services that talk together seamlessly no matter the provider, location or computational requirements.

Regarding this scenario, two specific initiatives appear to address separate parts of the puzzle: on one side, the ETICS project contributes to enhance the Grid software development process by creating the unified tool that implements recognised standards and best practices in order to guide the developer across a well-defined quality assurance process; on the other side, the OMII-Europe project, as part of the OGF initiative, aims to deliver production quality middleware relevant to large-scale and smaller collaborative Grids across the ERA by re-engineering key components to adapt them to interoperability standards: the OGSA-BES recommendation for Grid job submission. These two projects define the scope where the results obtained and presented across this document are focused.

As stated before (section 4.2), the general purpose of the work is to provide automatically interoperability testing based on standard compliance testing. And to face up the issue, two different goals have been successfully achieved: development of the compliance tests for the OGSA-BES recommendation and development of the testbed infrastructure where putting all together and performing the test.

The development of the compliance tests has required the analysis and design of the suite of unit and functional tests to ensure the compatibility

with the corresponding OGF recommendation [23]. This development has been integrated into the ETICS build tool what has greatly helped to understand the underlying complexity of the solution and to abstract the release process for different platforms. It has also been useful to facilitate the later integration of the tests execution in the testbed, managed by the ETICS facility.

The development of the testbed has been focused on the current status of the ETICS development and on the target services to be considered for the test. The integration of virtualization techniques (based on paravirtualization) has solved the big problem of installing normal services, that is, services prepared to be run on a normal machine in terms of, for instance, ports, paths in the file system, etc. With virtualization, the process of installing the IUT has been simplified merely to the installation itself of the services. From the point of view of the developer this is not so important because it must be something transparent, provided by the infrastructure. But from the point of view of the infrastructure maintainer, there is huge difference in deploying the IUT in a normal machine and deploying it under a virtual one. In the second case, the machine is not lost in case of failure, because the image can be easily removed and the space scratched using some monitoring tool (i.e. a watchdog). Regarding the performance, usage of virtual machines only penalizes the global performance when the target performance of the test is comparable to such penalty. For OGSA-BES tests, this aspect is out of scope.

Another important improvement is the integration in the ETICS information tree. While this tree contains all the information during the build part, extending such information for the test part provide an uniform view of the project, helping to see the global structure and status of it. In this sense, the design of general deployment modules for the OGSA-BES tests using the concepts of node and deployment enables the reutilization of the structure when the tests target on different distributions, for instance, gLite or UNICORE.

Despite the specific results of the test shown in the table 5.1 are all unsuccessful, just the ability to perform such test fulfills widely the initial objectives because it can be demonstrated that the IUT (gLite CREAM-BES development in that case) is still incomplete. It must be noticed that the fact that results of the compliance tests are all negative does not strictly mean the implementation is not compliant with the specification. In such cases, like in the current release of gLite CREAM-BES, the accuracy of the results is null because the service is not stable enough to be validated.

Therefore, besides direct results from this test execution, an important conclusion demonstrated within this work is that the quality in the devel-

opment process of some service does not rely only on its conformance with the underlying standards, its suitability to its purpose. Together with its correctness appears its stability, maintainability and final integration into the target structure. This concrete case has shown that to have compliance testing in place for the OGSA-BES service is not enough to assure the production quality stage of the software for final release. It has been its integration in a real use case (i.e. its installation, configuration and deployment) what has allowed to detect the weakness of the component: is not designed to be automatically or unattendedly deployed.

To know this kind of information in earlier stages of the deployment and keeping track of it is one of the key points to achieve successful results in complex distributed development projects like Grid ones. Usage of techniques like continuous integration in short life cycles enables to detect troubles, wrong designs, bottlenecks or incompatibility issues. As stated during the whole text and demonstrated in the last part, the ETICS project, in the design of its facility, exactly focuses on these topics and aids the whole range of users involved in the process to achieve the final goal: perceivable quality in the software.

Issues

The work described on this document has been performed in a high-end research environment. The cutting-edge state of the technologies used (e.g. the Grid, WS or virtualization) usually involves instability in the software components and unavailability in the features that delays and makes more difficult to reach the goals. The two major issues found concerns the Grid development and the evolution of current technologies.

Although the Grid paradigm is not so new and many investments have been done around the world, many questions still raise considering the best approach. As a result, the Grid community tries to understand all the consequences and has given many different answers. In addition, the large expectations created around it as the future solution for improve the efficiency of the current computing resources has driven to the current situation where many institutes develop its own solution. The community has identified this weakness and several international forums appeared to integrate existing solutions and promote standards or common practices that sum efforts.

In the case of the OGSA-BES recommendation, the distributed development of the service is a not so easy task to coordinate in order to follow a quality assurance process. To depend on the work of separate and far teams supposes to be in contact almost every moment and it is easy to

find misunderstandings or discrepancies between teams what at the end is translated into bad designs and delays.

On the side of the new technologies, the constant evolution of those ones makes very difficult to follow them properly. Sometimes, the life cycle of some technologies is shorter than their implementation. It is the case of WS. Initially, they were oriented as a mechanism to loosely couple services to their physical implementation, abstracting the specific details. This has been shown very powerful and extensible so now they are expanding so fast that a huge amount of recommendations has been produced. The definition of standards for WS interoperability, security, auditing, sessions, eventing, etc. improves its global functionality but also includes a dynamic component difficult to manage with when designing a service that interoperates with others.

Again, this is the case of the OGSA-BES in some way. By itself, it only defines the interface to communicate with a LRMS, but the evolution of other services in the Grid, like the security mechanisms, imposes additional features to the services that implements the recommendation. Dealing properly with this volatility is very important in order to achieve success in time and needs to define adequate development plans, considering the evolution of the technology in medium term.

What is the next?

The proof of concept developed across this document settles the basis where to establish the mechanisms to enforce usability, maintainability and interoperability in the development of distributed services.

After creating the facility to reproduce easily production scenarios where perform different kind of end-to-end tests, the next step is to improve it to make it more usable, intuitive and flexible, allowing the development of not only Grid services but any kind of software that requires to distribute the work across different teams.

Specifically, the OGSA-BES compliance tests must be extended to other implementations and check effectively their interoperability (based on design and not on interoperation), generating interoperability matrices between clients and services.

Based on this model, other recommendations may be implemented as a compliance test and integrated in the facility to increase the number of services that can be automatically tested for their final usage. The final objective would be to integrate this facility in the Grid, harnessing the experience and richness of its design to provide more intelligence to the

process, saving time and efforts. Some day the powerfulness of the Grid will be so accessible that it will be able to manage, maybe self manage, its own development.

Appendix A

Test implementation

A.1 ETICS commands

Listing A.1 shows the sequence of commands must be executed to launch the ETICS build for the compliance suite package.

Listing A.2 lists the set of commands that start a test activity. In this case, the listing shows the execution of the compliance test (process performed by the user).

Listing A.3 exemplifies the ETICS messaging service. The typed commands create the common session for all the parts involved in the test and show the usage of such session, publishing and consuming information.

Listing A.1: ETICS build sequence

```
etics-workspace-setup
etics-get-project gridtestbed
etics-checkout -c eu.omii.bes.compliance_R_0_2_0 \
    eu.omii.bes.compliance
etics-test -c eu.omii.bes.compliance_R_0_2_0 \
    eu.omii.bes.compliance
```

Listing A.2: ETICS test sequence

```
etics-workspace-setup
etics-get-project gridtestbed
etics-checkout -c eu.omii.compliance.glite \
    eu.omii.compliance
etics-test -c eu.omii.compliance.glite \
    eu.omii.compliance
```

Listing A.3: ETICS messaging service

```
etics-workspace-setup
etics-coschedule-local-setup -o /tmp/etics.uuid 4
7974cbf4-8677-4262-995e-e3365ef68f8c
etics-set --uuid 'cat /tmp/etics.uuid' \
    hostname etics.cern.ch
Done!
etics-get -b --uuid 'cat /tmp/etics.uuid' \
    hostname
etics.cern.ch
```

A.2 Testsuite design

Table A.1 details the full list of performed tests to verify the compliance of a service with the OGSA-BES recommendation.

Table A.1: Test methods description

Name	Description
BES-Factory	
CreateActivity	
1 BadFormattedJSDLActivityDocument	Sending a job which doesn't contain the JobDefinition tag
2 UnsupportedJSDLActivityDocument	Sending a job which contains too much information (i.e. features currently not supported)
3 UnsupportedFeatureActivity-Document	Sending a job which appends an additional desired description of the job not supported by the current BES implementation
4 PrivilegedActivity	Sending a job which requires additional privileges to run
5 StoppedService	Sending a job while the service is formally stopped
6 InvalidRequest	Sending a job with inconsistent information
7 NormalActivity	Sending a full job fulfilling all normal requirements
8 ActivityDocumentInResponse	Checking the presence of additional ActivityDocument in the response
GetActivityStatuses	
9 ZeroEPRAActivityIdentifier	Sending a zero-sized vector of EndPointReferences to get the status of the job
10 HundredEPRAActivityIdentifier	Sending one hundred EndPointReferences to check coherence in the results obtained
11 SingleResponseDescription	Sending one properly configured job and getting either description or the corresponding fault
12 SingleResponseFault	Sending one job with special privilege requirements in order to get a faulty description. Looking for NotAuthorizedFault

Continued on next page...

Table A.1 – Continued

13	UnknownEPRActivityIdentifier	Sending one random activity identifier to get not an exception but a error fault inside the answer. Looking for InvalidActivityIdentifier-Fault
14	InvalidRequest	Sending a request with inconsistent information
15	StatusTransitioning	Sending an empty job and checking that all status flow is implemented
16	NormalStatus	Sending a normal job
17	SubstatePresence	Sending a normal job and getting sub-states
TerminateActivities		
18	InvalidRequest	Sending inconsistent information in the request
19	ZeroEPRActivityIdentifier	Sending a zero-sized vector of EndPointReference
20	HundredEPRActivityIdentifier	Sending a hundred-sized vector of EndPointReferences
21	UnknownActivityIdentifier	Sending a well-formatted EndPointReference but pointing to non-existent activity
22	UnprivilegedRequest	Sending a request to perform the terminate action over a job for which additional privileges are needed
23	InTerminatedState	Sending a normal request and checking that the system moves to TerminatedState in case of successful request
24	IdempotentProperty	Sending two termination requests consecutive and checking that the second one has no effect (effect means change the status or throw any error)
25	NonChangesWhenFault	Running previously both methods (EPR not found and Non stoppable) and checking that there is no change in the status of the job
26	NonStoppableActivity	Sending a terminate request for a job that currently cannot be stopped

Continued on next page...

Table A.1 – Continued

<u>GetActivityDocuments</u>		
27	InvalidRequest	Sending inconsistent information in the request
28	ZeroActivityIdentifier	Sending a zero-sized vector of EndPointReference
29	HundredActivityIdentifier	Sending a hundred-sized vector of EndPointReferences
30	UnknownActivityIdentifier	Sending a well-formatted EndPointReference but pointing to non-existent activity
31	UnprivilegedRequest	Sending a request to perform the terminate action over a job for which additional privileges are needed
32	NormalRequest	Sending a document request and getting something that could be different from what it was sent
<u>GetFactoryAttributesDocument</u>		
33	InvalidRequest	Sending inconsistent information in the request
34	NormalService	Sending a normal request and checking the proper result: first and only one entity in the response, FactoryAttributesResourceDocument
<u>StartAcceptingNewActivities</u>		
35	StartAcceptingNewActivities	Try to invoke the starting command
<u>StopAcceptingNewActivities</u>		
36	StopAcceptingNewActivities	Try to invoke the stopping command

BES-Management

Appendix B

Screenshots

B.1 Modelization in ETICS

Figure B.1 shows the module tree designed for the compliance test (build and test parts) within ETICS. It shows the build modules (*eu.omii.bes*), the deployment module (*eu.omii.compliance*), the nodes (*eu.omii.compliance.-ce*, *eu.omii.compliance.client*, etc.) and the services and tests (*eu.omii.-compliance.glite.ce* and *eu.omii.compliance.glite.client*).

As mentioned, ETICS allows to send jobs through the CLI (listings ?? and A.2) or the WA. Figure B.2 shows precisely a resume of the web application interface for submitting test jobs.

B.2 Results

The following pictures show the results of deploying the OGSA-BES compliance test for the gLite CREAM-BES service within the ETICS facility.



Figure B.1: Compliance test tree within ETICS



Figure B.2: ETICS job submission



omineurope
OMII - Open Market Information Interface



Powered by
ETICS and
SIP NIMI

Module Summary

Project name: gridtestbed

Module name: eu.omii.compliance

Description: OGSABES compliance testing for glite

Version: 0.1.0

Release: 0

Vendor: OMII-Europe

License:

Execution Summary

Result: **Success**

Success rate: 100% (1/1)

Start time: 24/09/2007 17:36:21

End time: 24/09/2007 18:07:50

Duration: 00:31:29

Configuration: eu.omii.compliance.glite

Tag: HEAD

VCS Root: :pserver:anonymous@isscvcs.cern.ch:/local/repos/gridtestbed

Platform: slc4_i386_gcc346

Environment Properties

Build:

Test: etics-test -c
eu.omii.compliance.glite
eu.omii.compliance

ETICS Commands

Get project: etics-get-project
gridtestbed

Checkout: etics-checkout -c
eu.omii.compliance.glite
eu.omii.compliance



Copyright (c) 2007 ETICS

[Home](#)

[Logs](#)

[Package List](#)

Figure B.3: Compliance tests execution results page


Project name: gridtestbed
Project config: Unknown
Module name: eu.omii.compliance.repo
Module config: eu.omii.compliance.repo.glite
Build start time: 24/09/2007 17:43:15
Success rate: 100% (7/7)
Status: Success

Page generated at 24/09/2007 18:06:34
[Back to module overview page](#)

Component name	Configuration name	Last build time	Result
repo-janitor	repo-janitor v. 1.1	24/09/2007 17:43:16	Success
httpd	httpd v. 2.0.52-32.ent	24/09/2007 17:43:16	Success
httpd-deployment	httpd-deployment_cream	24/09/2007 17:43:26	Success
apt	apt v. 0.5.15cnc7	24/09/2007 17:43:26	Success
apt-deployment	apt-deployment_cream	24/09/2007 17:43:36	Success
eu.omii.compliance.glite.repo	eu.omii.compliance.glite.repo.HEAD	24/09/2007 17:44:34	Success
eu.omii.compliance.repo	eu.omii.compliance.repo.glite	24/09/2007 18:06:34	Success

Copyright (c) 2007 ETICS

Figure B.4: Repository deployment results page



Project name: gridtestbed
Project config: Unknown
Module name: eu.omii.compliance.wn
Module config: eu.omii.compliance.wn.gitle
Build start time: 24/09/2007 17:43:15
Success rate: 100% (2/2)
Status: **Success**

Page generated at 24/09/2007 18:06:24
[Back to module overview page](#)

Component name	Configuration name	Last build time	Result
eu.omii.compliance.gitle.wn	eu.omii.compliance.gitle.wn.HEAD	24/09/2007 17:43:15	Success
eu.omii.compliance.wn	eu.omii.compliance.wn.gitle	24/09/2007 18:06:24	Success

Copyright (c) 2007 ETICS

Figure B.5: Worker node deployment results page




Project name: gridtestbed
Module name: eu.omii.compliance.ce
Module config: eu.omii.compliance.ce.glite
Build status: Success
Description: OGSA-BES CE
Dependencies

Page generated at 24/09/2007 18:07:11
[Back to module list page](#)

Artefacts

Checkout / download log

```

09/24/07 17:41:06.170 INFO main [write] - Checking out configuration 'eu.omii.compliance.ce.glite'
09/24/07 17:41:06.171 INFO main [write] -
09/24/07 17:41:06.173 INFO main [write] - [checkout]: cvs -d :pserver:anonymous@isscvcs.cern.ch:/local/repns/gridtestbed co -A eu.omii.compliance.ce
09/24/07 17:41:06.175 INFO main [write] -
09/24/07 17:41:06.176 INFO main [_systemCall] - Calling system command: cvs -d :pserver:anonymous@isscvcs.cern.ch:/local/repns/gridtestbed co -A eu.omii.compliance.ce
09/24/07 17:41:06.780 ERROR main [write] - cvs checkout: Updating eu.omii.compliance.ce
09/24/07 17:41:07.013 INFO main [write] - U eu.omii.compliance.ce/deployCE.sh
  
```

Build log

No log available

Test report

```

09/24/07 17:43:15.896 INFO main [write] - Executing test: eu.omii.compliance.ce.glite
09/24/07 17:43:15.898 INFO main [write] -
09/24/07 17:43:15.932 INFO main [write] - Cannot found slccount. Plugin is disabled
09/24/07 17:43:15.951 INFO main [write] -
09/24/07 17:43:15.959 INFO main [write] - [init]: echo "Launching deployment of Glite CE"
09/24/07 17:43:15.962 INFO main [write] -
09/24/07 17:43:15.964 INFO main [_systemCall] - Calling system command: echo "Launching deployment of glite CE"
09/24/07 17:43:16.014 INFO main [write] - Launching deployment of glite CE
09/24/07 17:43:16.106 INFO main [write] - [test]: sh deployCE.sh -l "/root/wsetics/eu.omii.compliance.glite.ce" -d "--start" -r "--stop" -v
09/24/07 17:43:16.107 INFO main [write] -
09/24/07 17:43:16.109 INFO main [_systemCall] - Calling system command: sh deployCE.sh -l "/root/wsetics/eu.omii.compliance.glite.ce" -d "--start" -r "--stop" -v
09/24/07 17:43:16.162 INFO main [write] - Waiting for the repository...
09/24/07 17:45:15.980 INFO main [write] - found at lxb1107v2.cern.ch:8090
09/24/07 17:45:16.073 INFO main [write] - Publishing CE proposal for the WN (lxb1107v1.cern.ch)...
09/24/07 17:45:19.680 INFO main [write] - Done!
09/24/07 17:45:19.780 INFO main [write] - Waiting for the Worker Node...
09/24/07 17:45:33.667 INFO main [write] - found at lxb1303v2.cern.ch
Deploying CE (using lxb1107v2.cern.ch:8090 as repo and lxb1303v2.cern.ch as wn)...
09/24/07 17:45:33.778 INFO main [write] - Downloading certificates...
  
```

Figure B.6: CE deployment results page

Bibliography

- [1] D. Farber and K. Larson. *The Architecture of a Distributed Computer Systems Informal Description*. University of California, Irvine, CA, 1970.
- [2] G. Fernandez. *Curso de ordenadores. Conceptos básicos de arquitectura y sistemas operativos*. ETSIT Publication Service, Madrid, Spain, 2004.
- [3] K. Kesselman I. Foster and S. Tuecke. *The Anatomy of the Grid*. Technical report, Argonne National Laboratoty, University of Chicago, University of Southern California, 1999.
- [4] J. Nick I. Foster, K. Kesselman and S. Tuecke. *The physiology of the Grid*. Technical report, Argonne National Laboratoty, University of Chicago, University of Southern California, IBM Corporation, 1999.
- [5] K. Kesselman I. Foster, editor. *The Grid: Blueprint for a new Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.
- [6] I. Foster. *A globus primer*. Technical report, Globus Alliance, 2007. http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf.
- [7] EGEE JRA1. *EU Deliverable DJRA1.4: EGEE Middleware Architecture*. Technical report, EGEE Project, 2005. <https://edms.cern.ch/document/594698/>.
- [8] A. Delgado Peris F. Donno P. Méndez Lorenzo R. Santinelli A. Sciabà S. Burke, S. Campana. *gLite 3 User Guide*. Technical report, LCG Project, 2007. <https://edms.cern.ch/document/722398/>.
- [9] M. Romberg. *UNICORE: Beyond web-based job submission*. Technical report, Central Institute for Applied Mathematics, Research Center Jülich, 2000.

- [10] S. van den Berghe. UNICORE architecture and server components. Technical report, Grid Summer School, 2004.
- [11] Th. Lippert D. Mallmann R. Menday M. Rambadt M. Riedel M. Romberg B. Schuller Ph. Wieder A. Streit, D. Erwin. UNICORE: From Project Results to Production Grids. Technical report, John von Neumann-Institute for Computing (NIC), Forschungszentrum Jülich (FZJ), 2005.
- [12] A. Savva D. Berry A.Djaoui A. Grimshaw B. Horn F. Maciel F. Siebenlist R. Subramaniam J. Treadwell J. Von Reich I. Foster, H. Kishimoto. GFD.030: Open Grid Services Architecture. Technical report, Open Grid Forum, 2005.
- [13] H. Kishimoto J. Von Reich I. Foster, D. Gannon. GFD-I.029: Open Grid Services Architecture Use Cases. Technical report, Open Grid Forum, 2004.
- [14] R. Gibson. *Managing Computer Projects*. Prentice-Hall, 1992.
- [15] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1990.
- [16] B. Melton D. De Pablo A. Scheffer R. Stevens C. Mazza, J. Fairclough. *Software Engineering Standards*. Prentice Hall, 1994.
- [17] B. Beizer. *Software Testing Techniques*. International Thompson Computer Press, Boston, MA, 2nd edition edition, 1990.
- [18] E. van Veenendaal. *The testing practitioner*. UTN Publishers, 2002.
- [19] ETICS WP4. EU Deliverable D4.1: Requirements and specifications for unit, functional and regression testing. Technical report, ETICS Project, 2007. <https://edms.cern.ch/document/753643>.
- [20] Akogrimo WP2. Testbed description. Technical report, University of Honenheim, 2005.
- [21] F. Douglas C. Loosley. *High-performance client-server. A guide to building and managing robust distributed systems*. Wiley Computer Publishing, 1998.
- [22] M. Drescher D. Fellows A. Ly S. McGough D. Pulsipher A. Savva A. Anjomshoaa, F. Brisard. Job Submission Description Language (JSDL). Technical report, Open Grid Forum, 2005.

- [23] P. Lane W. Lee S. Newhouse S. Pickles D. Pulsipher C. Smith M. Theimer I. Foster, A. Grimshaw. OGSA Basic Execution Service. Technical report, Open Grid Forum, 2007.
- [24] S. Bradner. RFC2119: Key words for use in RFCs to Indicate Requirement Levels. Technical report, IETF: Network Working Group, 1997.
- [25] P. Wieder R. Yahyapour. GFD-I.064: Grid scheduling use cases. Technical report, Open Grid Forum, 2006.
- [26] I. Foster J. Frey S. Graham I. Sedukhin D. Snelling S. Tuecke W. Vambenepe K. Czajkowski, D. Ferguson. The WS-Resource Framework. Technical report, Globus Alliance, 2004.
- [27] R. Monzillo A. Nadalin P. Hallam-Baker, C. Kaler. WS-Security: X.509 Certificate Token Profile. Technical report, OASIS, 2007. <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0>.
- [28] Institute of Electrical and Electronics Engineers. *IEEE Computer Dictionary - Compilation of IEEE Standard Computer Glossaries*. IEEE, 1991.
- [29] Ting-Kau Leung D. Sidhu. Formal methods for protocol testing: A detailed study. In *IEEE Transactions on Software Engineering*, volume 15. IEEE, 1989.
- [30] A. Khodabandeh. Quality through Software Metrics. Technical report, Information and Programming Technology Group, CERN, 1998.
- [31] D. Johnson B. Jacobsen. Configuration management. Technical report, University of California, Berkeley, University of Colorado, Boulder, 1999.
- [32] M. Theimer G. Wasson M. Humphrey, C. Smith. JSDL HPC Profile Application Extension. Technical report, Open Grid Forum, 2006.
- [33] E. Stokes. Execution Environment and Basic Execution Service Model in OGSA Grids. Technical report, Open Grid Forum, 2007.
- [34] ETICS. ETICS overview: the ETICS build and test service. Technical report, ETICS Project, 2007.

- [35] ETICS WP5. ETICS User Manual. Technical report, ETICS Project, 2007. <https://edms.cern.ch/file/795312/1.0>.
- [36] ETICS WP3. EU Deliverable D3.2: Interoperability reports specifications, configuration, build and integration system. Technical report, ETICS Project, 2007. <https://edms.cern.ch/document/807014>.
- [37] ETICS WP4. EU Deliverable D4.5: Distributed test execution system. Technical report, ETICS Project, 2007. <https://edms.cern.ch/document/855714>.