Grant Agreement N° 215483

Title:	*Definition of the collaboration infrastructure and identification of appropriate tools for exchange of knowledge*
Authors:	*Domenico Laforenza (CNR), Frank Leymann (USTUTT), Nicola Tonellotto (CNR), Daniele Bocci (CNR), Gabriele Tolomei (CNR), Franco Maria Nardini (CNR)*
Editor:	*Domenico Laforenza (CNR)*
Reviewers:	*Ita Richardson (Lero-UL)*
	Jean-Louis Pazat (INRIA)
Identifier:	*Deliverable PO-IA-1.2.1*
Type:	*Deliverable*
Version:	*1*
Date:	*11 September 2008*
Status:	*Final*
Class:	*External*

Management Summary

This deliverable presents the concept of the S-Cube Pan-**European Distributed Service Laboratory** (**EDSL**) and its logical architecture. Among other objectives, S-Cube aims to integrate existing research infrastructures to provide access to a common shared facility for European institutions, thereby creating new opportunities for research, learning and experimentation. S-Cube beneficiaries will share resources (e.g. facilities, data, ICT infrastructures, etc.) across geographic and institutional boundaries. The shared resources will constitute the EDSL, which will enable the trial and evaluation of service concepts, technologies and system solutions. After a brief introduction to the EDSL concept, this document presents the EDSL logical architecture and a quite comprehensive state-of-the-art technologies and tools that might be evaluated and selected as potential "bricks" for the EDSL building.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany

Published S-Cube documents

These documents are all available from the project website located at http://www.s-cube-network.eu/

PO-JRA 1.1.1 State of the art report on software engineering design knowledge and Survey of HCI and contextual knowledge
PO-JRA-1.2.1 State-of-the-Art report on principles, techniques and methodologies for monitoring and adaptation
PO-JRA-1.3.1 Survey of quality related aspects relevant for SBAs
PO-JRA-2.1.1 State-of-the-art survey on business process modelling and Management
PO-JRA-2.2.1 Overview of the state of the art in composition and coordination of services

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community, which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-CUBE materials are available from URL: http://www.s-cube-network.eu/

Foreword

We wish to thank the following S-CUBE colleagues for their contributions, suggestions and criticisms: Schahram Dustdar (Vienna University of Technology), Frank Leymann (University of Stuttgart), Michele Mancioppi (Tilburg University), Andreas Metzger (University of Duisburg-Essen), JeanLouis Pazat (INRIA), Pierluigi Plebani (Politecnico di Milano), Fabrizio Silvestri (Consiglio Nazionale delle Ricerche).

Table of Contents

Table of illustrations

List of acronyms

ABI	Application Binary Interface
ACL	Access Control List
ACPI	Advanced Configuration and Power Interface
AGP	Accelerated Graphics Port
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BC	Binding Component
BIOS	Basic Input-Output System
BPEL	Business Process Execution Language
BSD	Berkeley Software Distribution
CIFS	Common Internet File System
CMS	Content Management System
CPAN	Comprehensive Perl Archive Network
CPU	Central Processing Unit
CSDMS	Collaborative Software Development Management System
CSS	Cascading Style Sheets
CVS	Concurrent Versions System
DPM	Disk Pool Manager
EAI	Enterprise Application Integration
EDSL	Pan-European Distributed Service Laboratory
EGEE	Enabling Grids for E-sciencE
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
FTS	File Transfer Service
GFAL	Grid File Access Library
GNU	GNU is Not Unix
GRUDU	Grid’5000 Reservation Utility for Deployment Usage
GSI	Grid Security Infrastructure
GUI	Graphical User Interface
GUID	Global Unique Identifier
HTTP	HyperText Transfer Protocol
HTTPS	HTTP over Secure socket layer
ICT	Information and Communication Technology

IDS	Intrusion Detection System
IIS	Internet Information Services
INRIA	Institut National de Recherche en Informatique et en Automatique
IP	Internet Protocol
IPSEC	IP Security
J2EE	Java 2 Enterprise Edition
JBI	Java Business Integration
JCA	Java Connector Architecture
JDBC	Java DataBase Connectivity
JDL	Job Description Language
JMS	Java Message Service
JMX	Java Management Extensions
JOnAS	Java Open Application Server
KAAPI	Kernel for Adaptive, Asynchronous Parallel and Interactive programming
L&B	Logging and Bookkeeping
LAN	Local Area Network
LCG	LHC Computing Grid project
LDAP	Lightweight Directory Access Protocol
LFC	Logical File Catalogue
LFN	Logical Filename
LHC	Large Hadron Collider
MPI	Message Passing Interface
MPLS	Multi Protocol Label Switching
MSW	Microsoft Windows
NFS	Network File System
OGF	Open Grid Forum
OS	Operating System
OSGi	Open Services Gateway initiative
P2P	Peer to Peer
POJO	Plain Old Java Object
POSIX	Portable Operating System Interface
QoS	Quality of Service
R-GMA	Relational Grid Monitoring Architecture
RCS	Revision Control System
RDF	Resource Description Framework
RENATER	Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche

RSS	Really Simple Syndication
SAAJ	SOAP with Attachments API for Java
SCM	Source Code Management
SE	Service Engine
SETI	Search for Extra-Terrestrial Intelligence
SMP	Symmetric MultiProcessing
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SQL	Structured Query Language
SRM	Storage Resource Manager
SSH	Secure Shell
SU	Service Unit
SURL	Site URL
SVN	Subversion
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TRANS	Transform – Route – Augment – Notify – Secure
TURL	Transport URL
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
VDT	Virtual Data Toolkit
VFS	Virtual File System
VLAN	Virtual LAN
VM	Virtual Machine
VMM	Virtual Machine Monitor
VO	Virtual Organization
VOMS	VO Membership Service
VPN	Virtual Private Network
WAR	Web Archive
WLCG	Worldwide LCG
WM	Workload Manager
WMS	Workload Management System
WS	Web Service
WS-I	WS Infrastructure
WSDL	WS Description Language
WSRF	WS Resource Framework
WYSIWYG	What You See Is What You Get
XML	eXtensible Markup Language

XMPP	Extensible Messaging and Presence Protocol
XSLT	eXtensible Stylesheet Language Transformations
XSQL	XML SQL

1 Introduction

The S-Cube mission is "to establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution and help shape the software-service based Internet which will underpin the whole of our future society". However, current research activities are very fragmented and, as a result, each research community concentrates mostly on its own specific research techniques, mechanisms, methodologies and vocabularies, which are not fully shared or aligned to the other research groups.

Looking at research topics covered in service-based systems, it is possible to distinguish between three functional layers that identify three distinct communities: Grid, Software Engineering and Business Process Management. Each one of those addresses particular issues and investigates specific aspects of service-based system research area. In order to implement the S-Cube mission, a set of key objectives has been identified for solving research fragmentation and identifying and bridging research gaps.

Among other objectives, S-Cube aims to integrate existing research infrastructures to provide access to a common shared facility for European institutions, thereby creating new opportunities for research, learning and experimentation. S-Cube beneficiaries will share resources (e.g. facilities, data, ICT infrastructures, etc.) across geographic and institutional boundaries. The shared resources will constitute the S-Cube **Pan-European Distributed Service Laboratory** (EDSL), which will enable the trial and evaluation of service concepts, technologies and system solutions. This sort of virtual laboratory network will facilitate close collaboration between researchers belonging to the communities previously mentioned, establishing a high-level distributed infrastructure to test, integrate, evaluate and benchmark service-based emerging technologies, methodologies and tools.

The EDSL will also support the communication within and across research communities by providing a common interface for access of resources, including intermediate results of the joint research activities, computing resources and infrastructure. In this role, the EDSL does not overlap the S-Cube Web Portal, that will be the official interface to the Internet, but it will offer an internal service to the S-Cube community.

Finally, it is worth noting that the building of new systems or middleware from scratch is out-of-scope of this project. In fact, as clearly reported in the S-CUBE Annex I, in the section(s) describing the activities of the WP-IA-1.2 (Pan-European Distributed Service Laboratory), WP-IA-1.2 will be devoted to establish a distributed laboratory infrastructure to test, integrate, evaluate and benchmark service-based applications relevant emerging technologies, methodologies and tools. The following sections present a quite comprehensive state-of-the-art technologies and tools that in our view might be evaluated and selected as potential "bricks" for the EDSL building. It is not our intention to present them in deep detail but, instead, to introduce their main features/capabilities.

2 EDSL Logical Architecture

2.1 The EDSL concept in a nutshell

This section illustrates the concept of the European Distributed Services Laboratory (EDSL). Before presenting its logical architecture a typical scenario of usage of the EDSL is described in the following with help of Figure 1. Let's suppose that a researcher belonging to the S-CUBE consortium needs to have access to some resources (hardware and/or software) in order to conduct his experiments (e.g. running an application composed by several components/services). In the real life he needs to contact someone, in his organisation or outside it (e.g. a public/private resource provider), able to supply him the required resources. This operation might be a bit time-consuming and annoying operation that could involve several negotiation phases with the resource provider in order to be authenticated/authorised to use the mentioned resources. Only after that the researcher will be able to access the mentioned resource in order to deploy, execute and monitor his application. The access to resources can be operated by using manual (e.g. using `ssh`) or automatic procedures or specialised tools (e.g. `kadeploy`) made available by the resource provider.

In order to make easier the researcher's life we propose to build an EDSL portal from which the researchers (in a simple, secure and transparent way) can: (i) to specify the required resources; (ii) to select them in a *plethora* of possible resources having the required characteristics; (iii) to deploy the application components/services on the selected resources; (iv) to start the execution, (v) to monitor the application behaviour.

Figure 1 depicts this situation. By using a common web browser the researcher accesses the EDSL portal and, after an authentication/authorisation phase, he is prompted with a screen presenting him a set of "islands" representing existent "resource providers" or "testbeds" (e.g. Grid5000, Planetlab,). Each island might be composed by several heterogeneous resources (e.g. machines, operating systems, middleware, applications, services,). Depending on his requirements (e.g. the need of a particular set of resources) the user can select one or more islands on the base of some general features characterising the selected island(s). For example, let us suppose that the researcher has to run a scalable application (e.g. an application composed by a huge number of software components/services). For that he requires a high number of machines to be available. Depending on the operational requirements of the software components (e.g. the need to have different operating system, installed in different clusters of machines) the researcher might ask that the required machines should be clustered by operating system type (e.g. 10 machines equipped with Windows, 5 machines with Mac OS X, 20 machines with Linux,). Since each island belongs to different organisations, e.g. Grid5000 (https://www.grid5000.fr/) is managed by the INRIA, the S-CUBE portal will require the researcher to ask the required authorisation/authentication rights to the owner organisation (the INRIA in this case) permitting the S-CUBE researcher to be authenticated/authorised once, only at the moment he accesses the portal. Then, the user will be enabled to access the resources belonging to the selected island(s). He could require deploying the software components on each machine made available in the island. This operation can be done manually (e.g. using the `ssh` command), or automatically (e.g. using some software tool offered by the chosen environment). When all the software components are deployed on the testbed, the researcher should be able to start their execution and monitor them, with tools provided by the run-time environment, in order to control the correct behaviour of his application.

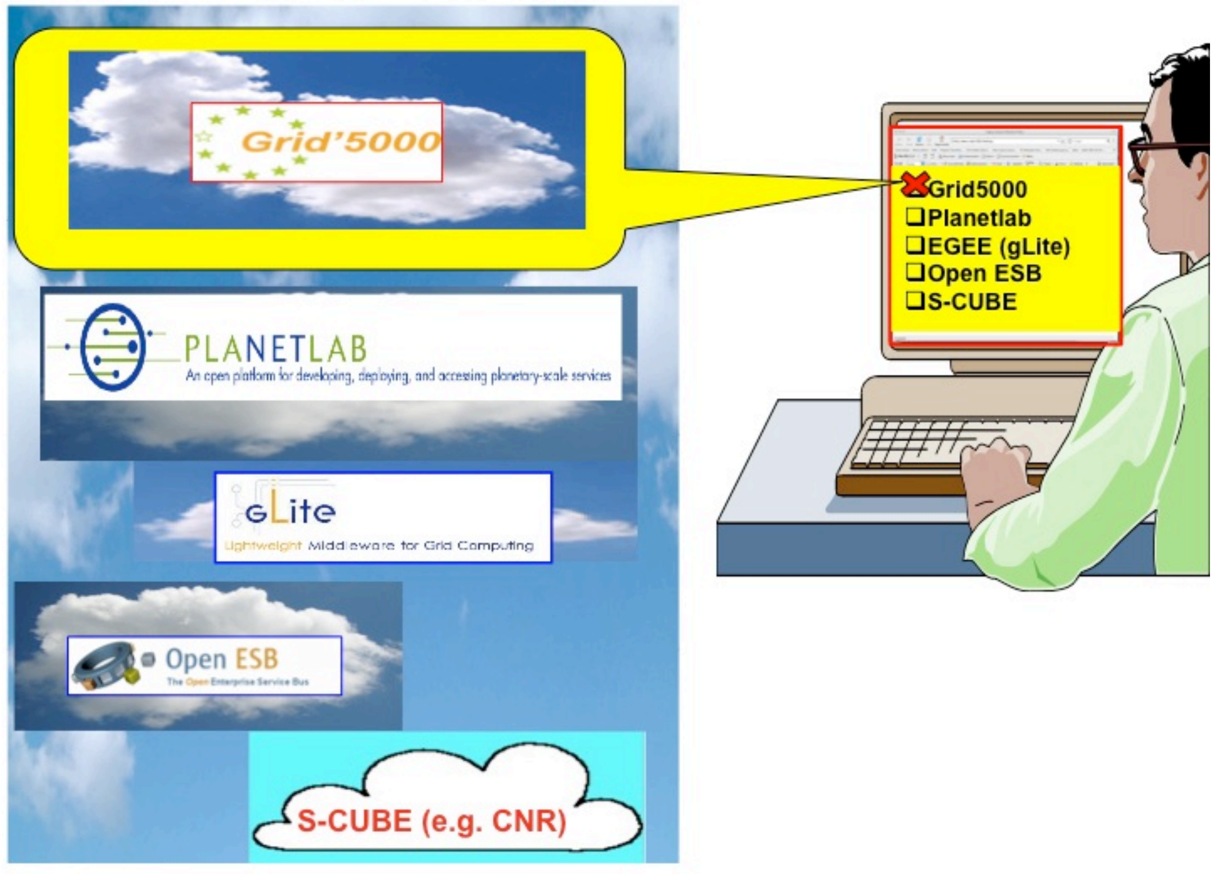
Grid'5000
PLANETLAB
An open platform for developing, deploying, and accessing planetary-scale services
gLite
Lightweight Middleware for Grid Computing
Open ESB
The Open Enterprise Service Bus
S-CUBE (e.g. CNR)
Grid5000
Planetlab
EGEE (gLite)
Open ESB
S-CUBE

Figure 1: A logical view of the EDSL Portal

As above mentioned the user will select the island that is more appropriate to run his application. To do that he should be aware about the features/capabilities offered by each island. The list of the main features/capabilities characterising each island is described in more detail in the following sections dedicated to the testbeds currently under evaluation. Obviously, the selected island(s) will depend by the real user requirements that will drive the selection of the most appropriate island (testbed) to use. Figure 1 shows also an island denoted as "S-CUBE (e.g. CNR)" that represents a very simple testbed, composed for instance, by some PCs at the disposal of several S-CUBE partners, equipped with de facto standard and open source software tools (e.g. Apache Tomcat). This environment might be enriched with some automatic and easy to use tool for software components/services deployment, execution and monitoring.

2.2 EDSL as a layered architecture

EDSL is intended as an effective and innovative test bed for the S-CUBE people on whom to test innovative scientific ideas on distributed services. It should be able to offer the best of the two following worlds: Grid and Service Oriented Computing, in order to demonstrate if and why the SOC and Grid approaches are really different and/or they compensate themselves.
EDSL can be seen as a layered architecture. Figures 2, 3 and 4 depict each layer of the EDSL logical architecture. Figure 2 shows the topology of the layer 0 (the "connecting tissue"); the edges (the lines) connecting the S-CUBE partners' sites are intended as "logical connections". This means that the S-Cube users will connect on a "best effort" basis using Internet.

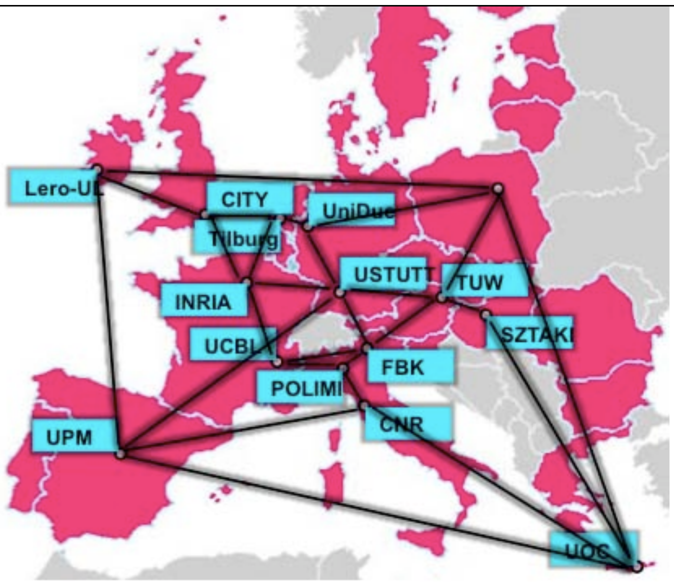

Figure 2: The logical topology of the EDSL (Layer 0)

Figure 3 sketches the EDSL layer 1 as a universe of "federated islands". Each island represents a "virtual organization" composed by several resources and characterized by well-established rules and policies regulating the access and usage of them. As reported in Figure 3, we can envisage the co-existence of several islands (e.g. Grid5000, EGEE/Glite) characterized by different level of operational complexity or affinities with the user's application. For example, a user interested in large scalable distributed platform might be interested to exploit the French Grid5000 testbed or the European EGEE/Glite production Grid platform (details on those platforms are reported in the following Section 3.1.1. and 3.2.1 respectively). The islands, often born independently and for different purposes, do not have explicit mechanisms to interoperate. Hence these offer their services without reciprocal knowledge. The user who wants to use them must choose explicitly the platform to use.
This choice depends from several factors, e.g. the complexity of the application, the availability of specific tools for the deployment/execution/monitoring, etc. Those islands can interoperate, permitting more sophisticated experiments, or not, but this is not an already available feature. This could be a target of some research activity developed inside the S-Cube project.

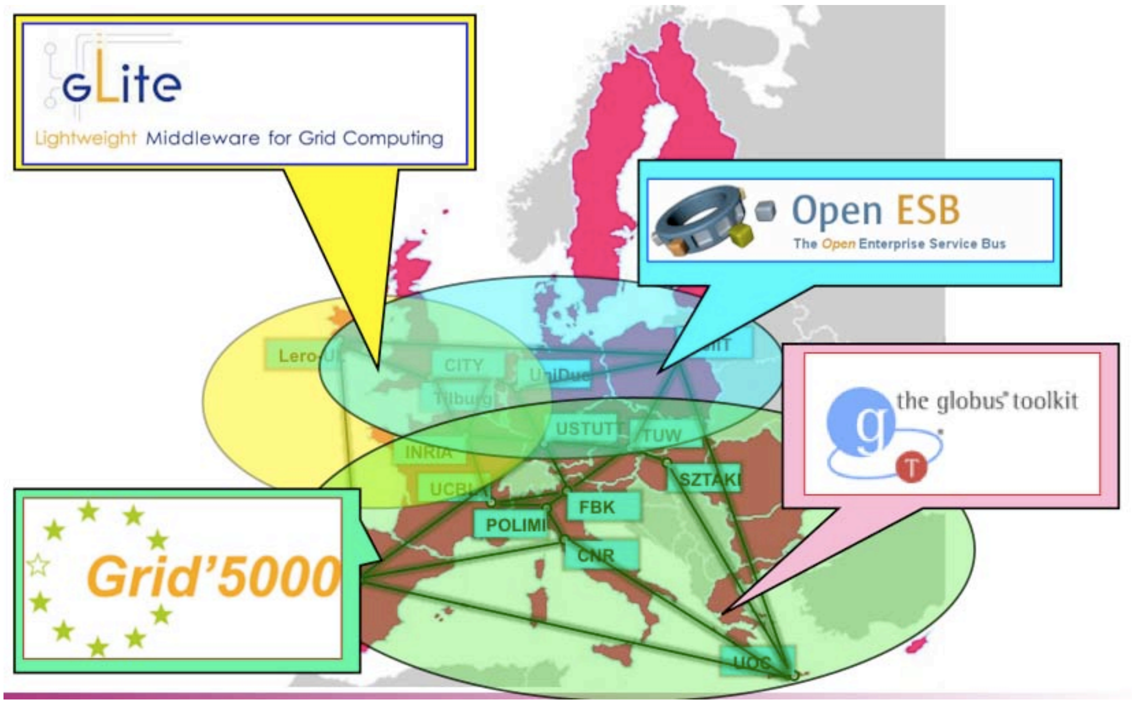

Figure 3: The EDSL as a universe of "federated islands" (Layer 1)

EDSL will have also tools of integration permitting a better collaboration among the S-CUBE partners. As said above the EDSL would be an integration enabler among the S-CUBE partners. In order to do that several open source de facto standard tools need to be exploited. For example, as depicted in Figure 4, a Wiki and a CVS system need to be chosen and deployed on top of the EDSL collaboration layer 2. Section 3.2 will discuss in deep detail the most important collaboration tools in order to guide the selection of the most suitable for adoption in the EDSL.

Figure 4: The collaborative tools of the EDSL (Layer 2)

2.3 *Steps towards a complete S-Cube EDSL implementation*

The EDSL view depicted in the previous sections 2.1 and 2.2 represents an asymptotical target to reach by the end of the S-CUBE projects. This means that during the project several steps towards a complete ESDL implementation will be done, taking in account the real requirements expressed by the EDSL users (e.g. the S-CUBE partners intended as the main beneficiaries of this infrastructure). Those requirements will be collected, organized and discussed in the deliverable CD-IA-1.2.2 (Detailed requirements analysis for a Distributed Services Laboratory Network) due at Month 9. In this deliverable both functional requirements for the EDSL and non-functional requirements will be detailed.

Figure 5 shows the steps that will be conducted in order to reach a complete view of the EDSL. We envisage the following steps:

- **Step#1:** Definition of the EDSL infrastructure and identification of appropriate tools [M6];
- **Step#2**: Detailed requirements analysis for the EDSL [M9];
- **Step#3**: Creation of the first "island" of the EDSL: the "S-CUBE island" [M14];
- **Step#4**: Integration of a second "island" in the EDSL: the Grid5000 island [M20]
- **Step#5**: Integration of a third "island" in the EDSL: the ESB island [M26]

At the successful completion of those steps the following actions will be started:

- A NoE-wide deployment of EDSL and the public roll out;
- Sustainability plans and outreach for EDSL.

Hence, the building of EDSL is intended as an incremental process that will start taking into account the real requirements of the users. The next section will describe in more details the Step#3 (Creation of the first "island" of the EDSL: the "S-CUBE island") that was specifically required by some S-CUBE partners in order to build the first "embryo" of a collaborative environment for starting tests and experiments on distributed services on the basis of their current requirements. Those initial requirements might hopefully change on the way depending also by the features offered by the EDSL that might offer to the users new potentialities for advanced experiments not foreseen at the beginning of the project. In order to address them, new "islands" will be incrementally included in the EDSL (e.g. a Grid5000 island) that will extend the features of the EDSL testbed allowing new kind of experiments.

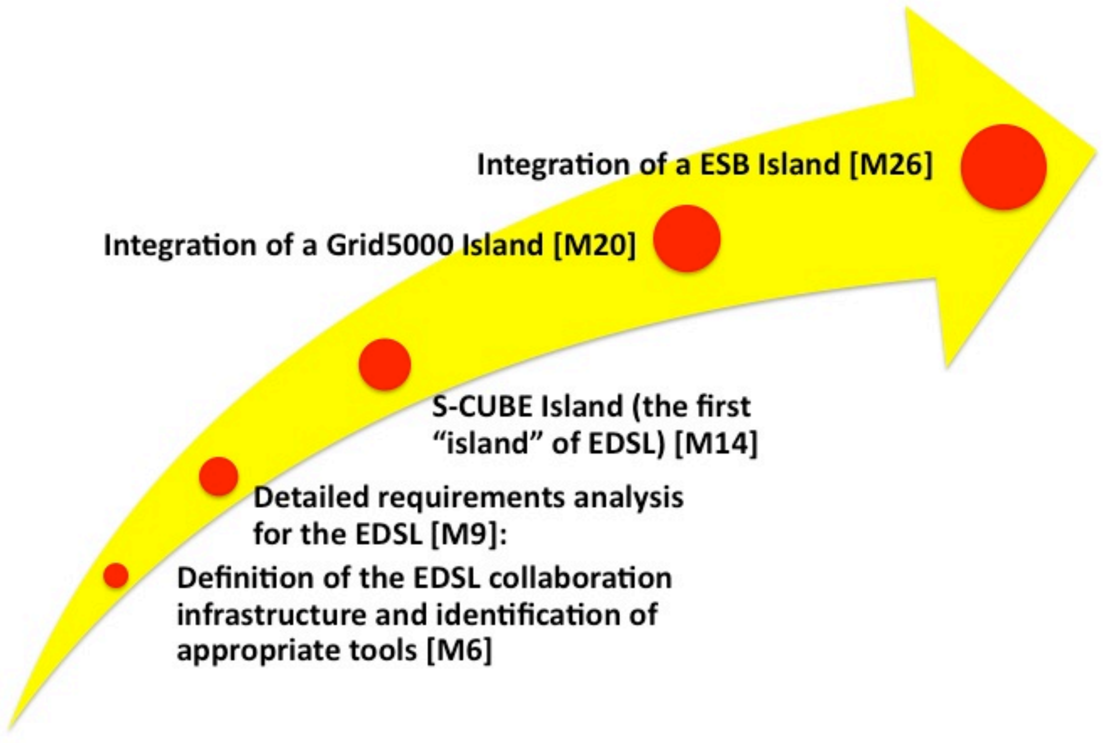
Integration of a ESB Island [M26]
Integration of a Grid5000 Island [M20]
S-CUBE Island (the first "island" of EDSL) [M14]
Detailed requirements analysis for the EDSL [M9]:
Definition of the EDSL collaboration infrastructure and identification of appropriate tools [M6]

Figure 5: Steps towards a complete EDSL implementation

2.4 S-Cube island: the first "island" of EDSL

All the "islands", which will be described in detail in the following sections, can only partially satisfy S-Cube partners' requirements. As an example, a scalability experiment could require a large number of hosts, and the Grid5000 platform could perfectly provides such capability, but at the same time there could be the need of an open-Internet infrastructure that Grid5000 can not offer.

For this reasons, to enable the EDSL we foresee another "island" designed to be the ad-hoc environment for the S-Cube partners. In designing this new "island", we want to keep the architecture as simple and flexible as possible, to allow an easy adaptation of the S-Cube internal requests. State-of-the-art and future specific tools will satisfy more specific requirements..

In our vision, the basic infrastructure of that "island" should be built on top of a hardware substrate composed by machines and network connections provided by partners that have them available. The infrastructure should be designed to allow the sharing of distributed resources due to the geographically distributed nature of partners. Each resource provided by S-Cube partners should be easily included within the infrastructure without any relevant effort. Moreover each partner will contribute in different way to the distributed laboratory.

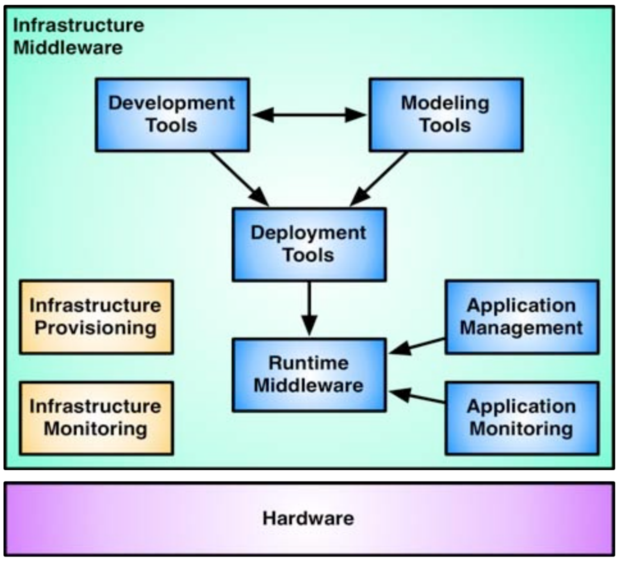

Figure 6: The coarse-grained structure of the EDSL island

Figure 5 depicts the overall coarse grained structure of the EDSL island, as provided by S-Cube. The Hardware (purple), middleware (green) and base infrastructure features (orange) build up the foundation of an EDSL island. Basically, the EDSL foundation is the basis for other people to use the prototypes that the S-Cube partners will build or offer to S-Cube. Similarly, it is the basis for the S-Cube partners to perform "integration tests" on the prototypes builds to verify the concepts and mechanisms developed by the undertaken research.

These partner components are shown in blue. They are artifacts like a Web container, an application server, an ESB, an orchestration engine, composition tool, development tools. These components may come with specific deployment tools allowing the distribution of specific executables in this environment. It may also be that some components come with specific management or monitoring features.

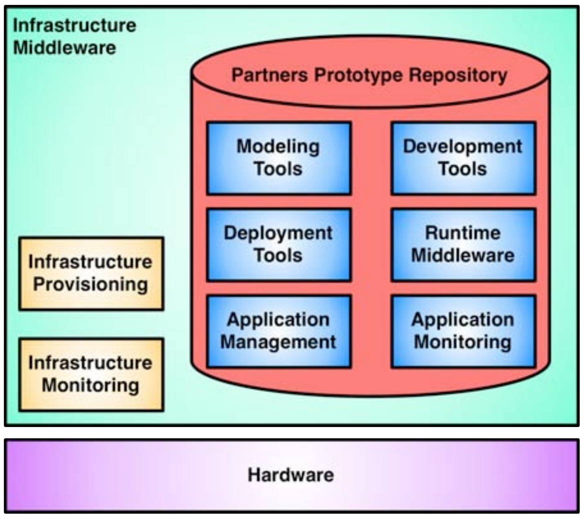

Figure 7: The operative view of the EDSL island

In order to build an EDSL island, the EDSL foundation will be coupled with a "repository" that other partners can populate with their prototypes, as shown in Figure 6. This repository is accessible by other people interested in figuring out what prototypes S-Cube offers. These people can also run the S-Cube prototype on the EDSL.

2.5 Island implementation Scenarios

In the following some implementation scenarios of an EDSL island will be presented.

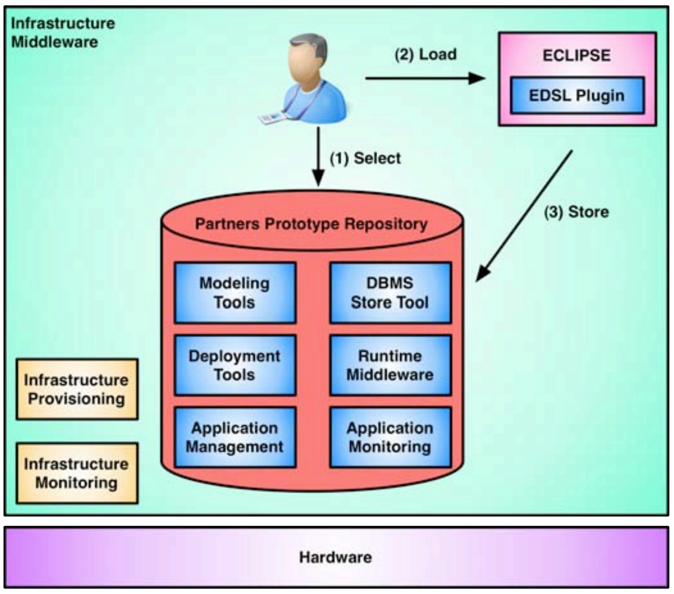

Figure 8: A simple implementation scenario

In Figure 7, the user (1) selects a prototype of a service composition tool stored in the EDSL repository. This tool is available as an Eclipse plugin. This plugin is (2) downloaded by the user and installed on the user local Eclipse platform. The artifacts manipulated using this tool can be stored (or exported) on a specific location provided by a resource manager of the EDSL infrastructure. Some of these artifacts could be (3) pushed inside the "prototypes repository" to make them available to other partners or for other tests.

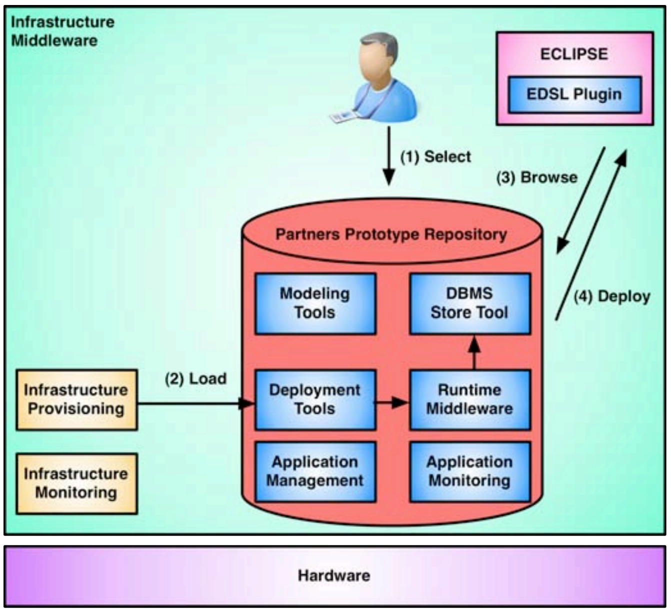

Figure 9: A complex implementation scenario

In Figure 8, a more complex example shows the exploitation of the infrastructure facilities in another EDSL island. The user (1) selects a runtime component from the "prototypes repository", which is needed to execute an application created by an S-Cube modeling or development tool. This runtime, together with the corresponding deployment tool, is (2) provisioned by the infrastructure features of the EDSL. After that, the user (3) browses the EDSL repository for his artifacts created before by an

S-Cube tool and selects some of them for deployment in the newly set up runtime. These artifacts are then (4) deployed via the formerly provisioned deployment tool specific for the provisioned runtime hosting the executable artifacts.

As a final example, partners could also provide “live” services instead of static resources. In order to evaluate new techniques and strategies for quality assurance or self-adaptation, using the EDSL, some partners can instantiate some benchmark services in favour of the whole S-Cube community. A user (1) accesses to a set of "benchmark" services and service composition tools (i.e., "real" application level services that can be composed to service-based applications); (2) deploys the service composition and the "implementation" of the techniques on the computational nodes by mean of deployment tools provided in the EDSL; (3) monitors the services in order to capture relevant data for the evaluation, using the previous instantiated benchmark service.

Of course, we must always take into account that not all experiments are possible on any platform.

3 State-of-the-art Technologies and Tools

This section presents several state-of-the-art technologies and tools to be evaluated and selected as potential “bricks” for the EDSL building. In the past years, several distributed architectures have been proposed, implemented and successfully exploited to perform large-scale, high performance distributed experiments in several applicative domains. Some constituting elements of such architectures might be useful to S-Cube partners to perform their research and execute large experiments. These *infrastructural approaches* enable system integration capabilities at middleware level. Instead, the *collaborative approaches* are orthogonal to the infrastructural approaches, enabling the exchange of information and improving the collaboration experiences. Again, some important collaboration tools are presented.

3.1 Infrastructural Approaches

Infrastructural approaches are state-of-the-art methodologies, technologies and tools enabling resources management, systems integration, software deployment and all related capabilities implemented at the middleware layer.
The main approaches described in the following are: *Grid’5000*, *Xen*, *PlanetLab*, *EGEE gLite* and *ESB*.

3.1.1 Grid’5000

Large scale distributed systems are excessively complex to be evaluated using theoretical models or simulators. Such systems need be experimented on real size, real life experimental platforms. In order to prove the effectiveness of the results, the experimentations must be reproducible. Some project, like XtreemOS project that work on operating systems, requires experimentations on the whole software stack. In order evaluate the scalability, these platforms must offer thousands of computation nodes.

Grid’5000 represents a stable, large, useful middleware exploited by several French research institutions at production level. The tools implemented allow the execution of large-scale experiments and the sharing of a massive set of resources. A subset of these tools might be very useful to the S-Cube researchers in performing their experiments.

Grid'5000 [1] [3] is a French national project aims at building a highly reconfigurable, controllable and monitorable experimental grid platform. It is funded by various research institutions and by the French Ministry of Research & Education. The main objective of the project is to provide the community of Grid researchers with an experimental platform fully, easily and remotely configurable. Indeed the users can configure the platform with its favorite network protocols, OS kernel and distribution, middleware, runtimes and applications.

The platform is build on top of mixed hardware geographically distributed in 9 cities (Nancy, Bordeaux, Grenoble, Lille, Lyon, Orsay, Rennes, Sophia, Toulouse). Currently, it is composed of 1696 nodes, 3418 processors (4790 cores, AMD and Intel). The nodes within each cluster communicate through a LAN of Gigabyte Ethernet links. Clusters within the same geographic area use the same LAN. Communications between clusters are made through the French academic network (RENATER). The "standard" architecture is based on 2,5Gbit/s carrying IP, but the last upgrades of the network introduced a dark fiber infrastructure, allocating dedicated 10Gbit/s "lambdas" for specific research projects. With the dark fiber infrastructure, Grid'5000 sites will see each other inside the same VLAN (10Gbit/s). Grid'5000 uses both Myrinet and Infiniband network infrastructures.

Grid’5000 provides a set of software tools to allow easy experiment preparation, run and control.

- Basic Tools of the Grid’500 testbed
  a. OAR: is a resource manager (or batch scheduler) for large clusters. It allows cluster users to submit or reserve nodes either in an interactive or a batch mode.
  b. Kadeploy: is a fast and scalable deployment system towards cluster and grid computing. It provides a set of tools, for cloning, configuring (post installation) and managing a set of nodes. Currently it deploys successfully Linux, *BSD, Windows, Solaris on x86 and 64 bits computers.
  c. TakTuk: is a tool for deploying parallel remote executions of commands to a potentially large set of remote nodes. It spreads itself using an adaptive algorithm and sets up an interconnection network to transport commands and to perform I/Os multiplexing/demultiplexing. The TakTuk mechanics dynamically adapt to environment (machine performance and current load, network contention) by using a reactive work-stealing algorithm that mixes local parallelization and work distribution.
  d. GRUDU: the Grid’5000 Reservation Utility for Deployment Usage is a tool for managing Grid’5000 resources, reservations and deployments. It provides a user-friendly interface.
- Tools and software developed on to the Grid’500 testbed
  a. KAAPI: the Kernel for Adaptative, Asynchronous Parallel and Interactive programming is a C++ library allowing the execution of multithreaded computations with data flow synchronization between threads. The library is able to schedule fine-/medium-sized grain programs on distributed machine. The data flow graph is dynamic (unfold at runtime). Target architectures are clusters of SMP machines.
  b. Katapult: is a small, well-tested script to automatically start experiments using deployments. Most experiments start by deploying the nodes, re-deploying the nodes if too many of them failed, copying the user SSH key to the node, etc. Katapult automates all those tasks.
  c. Marcel: is a POSIX-compliant thread library featuring a programmable scheduler designed for hierarchical multiprocessor architectures.
  d. Mad-MPI: is an efficient implementation of MPI for fast networks.
  e. MPICH-Madeleine: is an MPI implementation for clusters and clusters of clusters with heterogeneous networks.
  f. NewMadeleine: is a communication library that provides extended capabilities for dynamic communication optimization on top of high performance networks.
  g. Wrekavoc: the goal of Wrekavoc is to define and control the heterogeneity of a given platform by degrading CPU, network or memory capabilities of each node composing this platform. The degradation is done remotely, without restarting the hardware. The control is fine, reproducible and independent (one may degrade CPU without modifying the network bandwidth).

Security

Grid'5000 infrastructure allows the users to deploy their specific software stack. This flexibility however does not ensure the correctness of security configuration and the machines are to be considered not protected. To enforce a sufficient protection on the resources, the Grid'5000 architecture provides an isolated network domain where communication fly without restrictions between sites and are not possible directly with outside world [2]. The isolated network domain rule ensures that the infrastructure will resist to hacker attacks and will not be used as basis of attacks (i.e. massive Denial-of-Service). The sites are interconnected in VPN with a combination of DiffServ (network layer) and MPLS technology (link layer) provided by RENATER. The use of the MPLS VPN for link layer provides better performance versus IP VPN (Ipsec). This solution provides, simultaneously with security, a guarantee required for reproducible experimental conditions and performance measurements. Moreover, a strong authentication and authorization check is done first to enter the lab and then to log in Grid'5000 nodes from the lab. The access uniformity into the whole platform has been fulfilled by the installation of an LDAP directory. But the reliability of the authentication system is critical. The structure of the authentication mechanism is designed in a distributed deployment, so that a local network outage should not break the authentication process on other sites. Hence, each Grid'5000 site manages its own user accounts and it runs an LDAP server containing the same tree: under a common root, a branch is defined for each site. The local administrator manages its own branch while the other branches are periodically synchronized from remote servers. This design is transparent to the final user: she can access any of the Grid'5000 sites or services (monitoring tools, wiki, deployment, etc.) with the same account. The user can get node access by ssh. The authentication is performed by means of ssh public key. This brings, together with a strong authentication, the single-sign-on feature between all nodes allocated to a user or between front-end nodes.

Data Management

Each user has a home directory where he can hold private data. However, the home directories are local to every site. They are shared on any given cluster through NFS, but distribution to another remote site is done by the user through classical file transfer tools (rsync, scp, sftp, etc.). Data transfers with the outside of Grid'5000 are restricted to secure tools to prevent identity spoofing and public key authentication is used to prevent brute-force attacks.

Node Reconfiguration

Grid'5000 provides a reconfiguration mechanism that allows users to deploy, install, boot and run their specific software images, possibly including all the layers of the software stack. In a typical experiment sequence, a user can reserve a partition of infrastructure nodes, deploy its software image, reboot all the machines of the partition, run the experiment, collect results and relieve the machines. This reconfiguration capability allows all researchers to run their experiments in the software environment exactly corresponding to their needs. The node reconfiguration operation is performed by a deployment tool suite called Kadeploy. This tool suite allows users to deploy their own software environment on a disk partition of selected nodes. The software environment can contain all software layers from OS to application needed by users for theirs experiments. Kadeploy allows the deployment of multiple computing environments on clusters or grids without compromising the original system/boot sector on every node. A database in the Kadeploy architecture contains the environment descriptions (kernel, initrd, custom kernel parameters, desired filesystem for environment, associated post-installation). When the user initiates a deploy operation, he provides an environment name allowing to retrieve associated information from the database. The user can use prepared environments or register its own environment. After the deployment completion, the target node reboots on the deployed system.

The commands provided by Kadeploy suite are:

- *kaaddnode*: registers nodes in deployment system;
- *kadelnode*: unregisters nodes in deployment system;

- *karecordenv*: registers an environment image in deployment system;
- *kaarchive*: creates an environment image;
- *kacreateenv*: creates and registers an environment image in deployment system;
- *kadeploy*: deploys an environment image;
- *kaconsole* : opens a remote console;
- *kareboot*: reboots according to requested reboot type.

Scheduling

The platform provides a simple way to experiment scheduling and resource allocation: this is managed by a resource management system called OAR at cluster level and by a simple broker at the grid level. OAR provides most of the important features implemented by other batch schedulers such as priority scheduling by queues, advance reservations, backfilling and resource match making. OAR has, compared to other tools, the following advantages:

- No specific daemon on nodes.
- No dependence on specific computing libraries like MPI. It supports all sort of parallel user applications.
- Upgrades are made on the servers, nothing to do on computing nodes.
- CPUSET (2.6 Linux kernel) integration restricting the jobs on assigned resources (also useful to clean completely a job, even parallel jobs).
- All administration tasks are performed with the taktuk command (a large scale remote execution deployment).
- Hierarchical resource requests (handle heterogeneous clusters).
- Gantt scheduling (so you can visualize the internal scheduler decisions).
- Full or partial time-sharing.
- Checkpoint/resubmit.
- Licenses server management support.
- Best effort jobs: if another job wants the same resources then it is deleted automatically (useful to execute programs like SETI@home).
- Environment deployment support (Kadeploy).

The commands provided by OAR suite are:

- *oarstat*: prints jobs in execution mode;
- *oarnodes*: prints information about cluster resources (state, which jobs on which resources, resource properties, ...);
- *oarsub*: allows a user to submit a job;
- *oardel*: is used to delete or checkpoint job(s);
- *oarhold*: is used to remove a job from the scheduling queue if it is in the "waiting" state;
- *oarresume*: resumes jobs in the states “hold” or “suspended”.

All large-scale operations like parallel tasks launching, nodes probing or monitoring are performed using a specialized parallel launching tool named Taktuk. At grid level, a simple broker allows co-allocating set of nodes on every selected cluster. The resource management system is coupled with node reconfiguration operation at different points. The requests for node reconfiguration are handled in a specific scheduling queue and, just while the deployment, the users are granted with rights for system images installation. After the completion of experiments, all nodes are rebooted in a default environment.

3.1.2 Xen

Recent advances in virtualization technologies enabling data centers to consolidate servers, normalize hardware resources, and isolate applications on the same physical server are driving rapid adoption of server virtualization in Linux environments.

Virtualization software abstracts the underlying hardware by creating an interface to Virtual Machines (VMs), which represent virtualized resources such as CPUs, physical memory, network connections, and block devices. Software stacks including the Operating System (OS) and applications are executed on top of the VMs. Several VMs can run simultaneously on a single physical server. Multiplexing of physical resources between the VMs is enforced by a VM Monitor (VMM), which is also designed to provide the required translations of operations from the VMs to the physical hardware. The S-Cube EDSL might exploit such functionalities to manage different configurations on a reduced set of computing resources.

Full virtualization vs. "Para-virtualization"

There are several ways to implement virtualization: two leading approaches are full virtualization and para-virtualization. Full virtualization is designed to provide total abstraction of the underlying physical system and to create a complete virtual system in which the guest operating systems can execute. No modification is required in the guest OS or application; the guest OS or application is not aware of the virtualized environment so they have the capability to execute on the VM just as they would on a physical system. This approach can be advantageous because it enables complete decoupling of the software from the hardware. As a result, full virtualization can streamline the migration of applications and workloads between different physical systems. Full virtualization also helps provide complete isolation of different applications, which helps make this approach highly secure. Microsoft Virtual Server and VMWare ESX Server software are examples of full virtualization.

However, full virtualization may incur a performance penalty. The VMM must provide the VM with an image of an entire system, including virtual BIOS, virtual memory space, and virtual devices. The VMM also must create and maintain data structures for the virtual components, such as a shadow memory page table. These data structures must be updated for every corresponding access by the VMs. In contrast, para-virtualization presents each VM with an abstraction of the hardware that is similar but not identical to the underlying physical hardware. Para-virtualization techniques require modifications to the guest OSs that are running on the VMs but keeping Application Binary Interfaces (ABIs). As a result, the guest OSs are aware that they are executing on a VM, allowing for near-native performance and applications running inside a guest OS has not to be modified. Para-virtualization methods are still being developed and thus have limitations, including several insecurities such as the guest OS cache data, unauthenticated connections, and so forth.

Xen 3.0 approach

Xen 3.0 is an open source virtualization software based on para-virtualization approach developed for x86 architectures, targeted at hosting up to 100 virtual machine instances simultaneously on a single modern server [6] [7] [8].

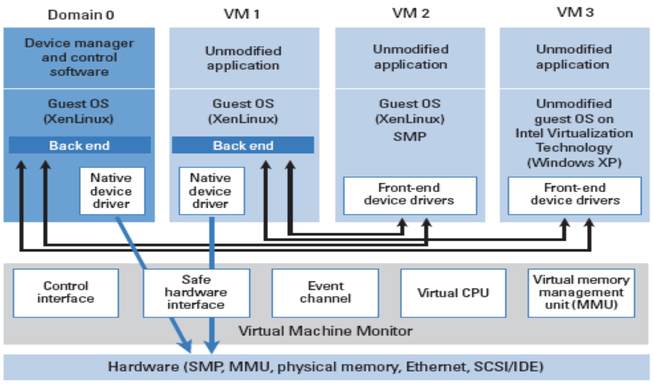

Figure 10: Xen 3.0 architecture: Hosting four VMs

The picture above shows the architecture of Xen 3.0 hosting four VMs: Domain 0, VM 1, VM 2, and VM 3. Each VM 1, VM 2 and VM 3 with its own guest OS is called a domain in Xen jargon. This architecture includes the Xen Virtual Machine Monitor (VMM), which abstracts the underlying physical hardware and provides hardware access for the different virtual machines. Only Domain 0 can access the control interface of the VMM, through which other VMs can be created, destroyed, and managed. Management and control software runs in Domain 0. Administrators can create virtual machines with special privileges (such as VM 1 in the picture) that can directly access the hardware through secure interfaces provided by Xen. Administrators can create other virtual machines that can access the physical resources provided by Domain 0’s control and management interface in Xen. In this example, the guest operating systems in VM 1 and in VM 2 are modified to run above Xen and also have Xen-aware drivers to enable high performance: near-native performance can be achieved through this approach. Anyway, unmodified guest operating systems are also supported.

In addition, the developers of Xen 3.0 plan to include support for virtual machines with symmetric multiprocessing (SMP) capabilities, 64-bit guest operating systems, Accelerated Graphics Port (AGP), and Advanced Configuration and Power Interface (ACPI). In a virtual data center framework, CPU, memory and I/O components need to be virtualized: Xen 3.0 is designed to enable para-virtualization of all these hardware components.

The Intel x86 architecture provides four levels of privilege modes. These modes, or rings, are numbered 0 to 3, with 0 being the most privileged. In a non-virtualized system, the OS executes at ring 0 and the applications at ring 3, while rings 1 and 2 are typically not used. In Xen para-virtualization, the VMM executes at ring 0, the guest OS at ring 1, and the applications at ring 3: this approach helps to ensure that the VMM possesses the highest privilege, while the guest OS executes in a higher privileged mode than the applications and is isolated from the applications. Privileged instructions issued by the guest OS are verified and executed by the VMM. Guest OSs use so-called hypercalls to invoke operations in Xen, following an approach that is similar to system calls between processes and OS in a classical single OS hosted non-virtualized environment.

In a non-virtualized environment, the OS expects contiguous memory. Guest OSs in Xen para-virtualization are modified to access memory in a non-contiguous manner and are responsible for allocating and managing page tables. However, direct writes are intercepted and validated by the Xen VMM.

In a fully virtualized environment, hardware devices are emulated. Xen para-virtualization exposes a set of clean and simple device abstractions. For example, I/O data to and from guest OSs is transferred using shared memory ring architecture (memory is shared between Domain 0 and the guest domain) through which incoming and outgoing messages are sent. Similar to hardware interrupts, Xen supports a lightweight event-delivery mechanism that is used to send asynchronous notification to a domain.

Modifying the guest OS is not feasible for non–open source platforms such as Microsoft Windows 2000 or Windows Server™ 2003 operating systems. As a result, such non-modifiable operating systems are not supported in a para-virtualization environment until processors will not provide additional instructions designed to support full virtualization (e.g. Intel VT or AMD V technologies).

Xen 3.0 management tools

Two important management tools in Xen 3.0 are xend and xm. The Xen daemon, xend, is a Python program that forms a central point of control for starting and managing VMs. Major functions include invoking setup of virtual networking for VMs, providing a console server, and maintaining the Xen events log. The Xen command-line interface, xm, provides functionality to create, destroy, save, restore, shut down, migrate domains, and so forth. It also enables administrators to configure the CPU scheduler, list active domains, adjust the memory footprint using ballooning, and call an xend HTTP application programming interface (API) directly.

3.1.3 PlanetLab

PlanetLab is an entry point onto a new Internet and supports a diversity of services, bringing together a range of new and known ideas and techniques within a comprehensive design. Clearly it shares several concepts and motivations with the S-Cube goal, and the S-Cube participants may reuse parts of the software developed by the PlanetLab partners.

PlanetLab could be defined in different ways, such as:

a) <u>A collection of machines distributed over the globe</u>

Research institutions host most of the machines, although some are situated in co-location and routing centers (e.g., on Internet2's Abilene backbone). All of the machines are connected to the Internet. One of the goals of PlanetLab is to grow to 1,000 widely distributed nodes that peer with the majority of the Internet's regional and long haul backbones.

b) <u>A software package</u>

All PlanetLab machines run a common software package that includes a Linux-based operating system, mechanisms for bootstrapping nodes and distributing software updates, a collection of management tools that monitor node health, audit system activity, and control system parameters, and a facility for managing user accounts and distributing keys. This software is distributed as a package, called MyPLC, which others can use to build and deploy their own "private PlanetLabs".
The key objective of this software is to support distributed virtualization - the ability to allocate a slice of PlanetLab's network - wide hardware resources to an application. This allows an application to run across all (or some) of the machines distributed over the globe, where at any given time, multiple applications may be running in different slices of PlanetLab.

c) <u>An overlay network testbed</u>

One of PlanetLab's main purposes is to serve as a testbed for overlay networks. Research groups are able to request a PlanetLab slice in which they can experiment with a variety of planetary-scale services, including file sharing and network-embedded storage, content distribution networks, routing and multicast overlays, QoS overlays, scalable object location, scalable event propagation, anomaly detection mechanisms, and network measurement tools. There are currently over 600 active research projects running on PlanetLab.
The advantage to researchers in using PlanetLab is that they are able to experiment with new services on a large scale under real-world conditions. The example services outlined above all benefit from being widely distributed over the Internet: from having multiple vantage points from which applications can observe and react to the network's behavior, from being in close proximity to many data sources and data sinks, and from being distributed across multiple administrative boundaries.

d) <u>A deployment platform</u>

In addition to supporting short-term experiments, PlanetLab is also designed to support long-running services that support a client base. That is, rather than view PlanetLab strictly as a testbed, it can be allowed a long-term view in which the overlay is both a research testbed and a deployment platform, thereby supporting the seamless migration of an application from early prototype, through multiple design iterations, to a popular service that continues to evolve. Using an overlay as both a research testbed and a deployment platform is synergistic. As a testbed, the overlay's value is to give researchers access to a:

- large set of geographically distributed machines;
- realistic network substrate that experiences congestion failures, and diverse link behaviors.

e) The potential for a realistic client workload

Its value as a deployment platform is to provide researchers with a direct technology transfer path for popular new services, and users with access to those new services. Supporting both roles is critical to the success of the system.

f) A microcosm of the next Internet

Not only are researchers evaluating and deploying end-user services on top of PlanetLab, but it is also arguable they develop foundational sub-services that can be folded back in to PlanetLab, thereby enhancing the facility for others. A long-term goal is to identify the common building block services upon which other services and applications can be constructed, or said another way, the goal is to understand how the Internet can be architected to better support overlays.
This perspective is motivated by the general question of how the networking research community can best impact the global Internet. Unfortunately, the very commercial success that has fueled our increased dependency on the Internet has also reduced our ability to evolve its underlying architecture to meet new demands and correct emerging vulnerabilities.
Overlay networks provide an opportunity to introduce disruptive technologies. Overlay nodes can be programmed to support the new capability or feature, and then depend on conventional nodes to provide the underlying connectivity. Over time, if the idea deployed in the overlay proves useful, there may be economic motivation to migrate the functionality into the base system, that is, add it to the feature set of commercial routers. On the other hand, the functionality may be complex enough that an overlay layer may be exactly where it belongs. The overarching goal is to support the introduction of disruptive technologies into the Internet through the use of overlay networks. PlanetLab is an essential element of this vision.

In Depth Dicsussion

PlanetLab is a geographically distributed overlay platform designed to support the deployment and evaluation of planetary-scale network services [9]. It currently includes over 350 machines spanning 150 sites and 20 countries. It supports over 450 research projects focused on a wide range of services, including file sharing and network embedded storage [10] [11] [12], content distribution networks [13], routing and multicast overlays [14] [15], QoS overlays [16], scalable object location services [17] [18] [19] [20], anomaly detection mechanisms [21], and network measurement tools [22].
As a distributed system, PlanetLab is characterized by a unique set of relationships between principals (e.g.: users, administrators, researchers, service providers) that make the design requirements for its operating system different from traditional hosting services or timesharing systems.
The first relationship is between PlanetLab as an organization, and the institutions that own and host Planet-Lab nodes: the former has administrative control over the nodes, but local sites also need to enforce policies about how the nodes are used, and the kinds and quantity of network traffic the nodes can generate. This implies a need to share control of PlanetLab nodes.
The second relationship is between PlanetLab and its users, currently researchers evaluating and deploying planetary-scale services. Researchers must have access to the platform, which implies a distributed set of machines that must be shared in a way they will find useful. A Planet-Lab “account”,

together with associated resources, must therefore span multiple machines. This abstraction is called a slice, and is implemented using a technique called distributed virtualization.
A third relationship exists between PlanetLab and those researchers contributing to the system by designing and building infrastructure services, that is, services that contribute to the running of the platform as opposed to being merely applications on it. Not only each of these services must run in a slice, but also PlanetLab must support multiple, parallel services with similar functions developed by different groups. This principle is called unbundled management, and it imposes its own requirements on the system.
Finally, PlanetLab exists in relation to the rest of the Internet. Experience shows that the experimental networking performed on PlanetLab can easily impact many external sites' intrusion detection and vulnerability scanners. This leads to requirements for policies limiting what traffic PlanetLab users can send to the rest of the Internet, and a way for concerned outside individuals to find out exactly why they are seeing unusual traffic from PlanetLab. The rest of the Internet needs to feel safe from PlanetLab.

Distributed Virtualization

PlanetLab services and applications run in a slice of the platform: a set of nodes on which the service receives a fraction of each node's resources, in the form of a Virtual Machine (VM). Virtualization and virtual machines are, of course, well-established concepts. What is new in Planet-Lab is distributed virtualization: the acquisition of a distributed set of VMs that are treated as a single, compound entity by the system. To support this concept, PlanetLab must provide facilities to create a slice, initialize it with sufficient persistent state to boot the service or application in question, and bind the slice to a set of resources on each constituent node. However, much of a slice's behavior is left unspecified in the architecture. This includes exactly how a slice is created, in the context of unbundled management, as well as the programming environment Planet-Lab provides. Giving slices as much latitude as possible in defining a suitable environment means, for example, that the PlanetLab OS does not provide tunnels that connect the constituent VMs into any particular overlay configuration, but instead provides an interface that allows each service to define its own topology on top of the fully-connected Internet. Similarly, PlanetLab does not prescribe a single language or runtime system, but instead allows slices to load whatever environments or software packages they need.

Isolating Slices

PlanetLab must isolate slices from each other, thereby maintaining the illusion that each slice spans a distributed set of private machines.

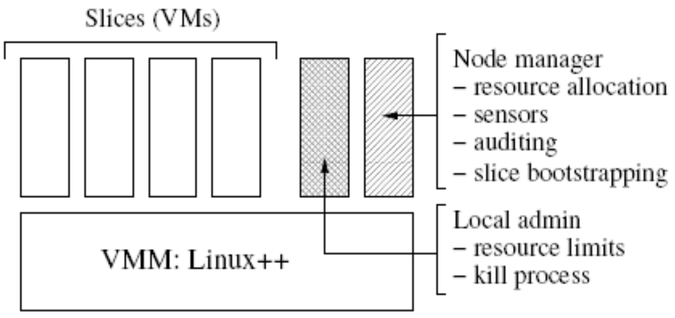

Figure 11: PlanetLab Node Architecture

The same requirement is seen in traditional operating systems, except that in PlanetLab the slice is a distributed set of VMs rather than a single process or image. Per-node resource guarantees are also required: for example, some slices run time-sensitive applications, such as network measurement services, that have soft real time constraints reminiscent of those provided by multimedia operating systems. This means three things with respect to the PlanetLab OS:

1) It must allocate and schedule node resources (cycles, bandwidth, memory, and storage) so that the runtime behavior of one slice on a node does not adversely affect the performance of another on the same node. Moreover, certain slices must be able to request a minimal resource level, and in return, receive (soft) real time performance guarantees.

2) It must either partition or contextualize the available name spaces (network addresses, file names, etc.) to prevent a slice interfering with another, or gaining access to information in another slice. In many cases, this partitioning and contextualizing must be coordinated over the set of nodes in the system.

3) It must provide a stable programming base that cannot be manipulated by code running in one slice in a way that negatively affects another slice. In the context of a Unix- or Windows-like operating system, this means that a slice cannot be given root or system privilege.

Resource scheduling and VM isolation were recognized as important issues from the start, but the expectation was that a “best effort” solution would be sufficient for some time. However, excessive loads (especially near conference deadlines) and volatile performance behavior (due to insufficient isolation) were the dominant problems in early versions of the system. The lack of isolation has also led to significant management overhead, as human intervention is required to deal with run-away processes, unbounded log files, and so on.

Isolating PlanetLab

The PlanetLab OS must also protect the outside world from slices. PlanetLab nodes are simply machines connected to the Internet, and as a consequence, buggy or malicious services running in slices have the potential to affect the global communications infrastructure. Due to PlanetLab’s widespread nature and its goal of supporting novel network services, this impact goes far beyond the reach of an application running on any single computer. This places two requirements on the PlanetLab OS.
It must thoroughly account resource usage, and make it possible to place limits on resource consumption so as to mitigate the damage a service can inflict on the Internet. Proper accounting is also required to isolate slices from each other. Here, it has to be evaluated both the node’s impact on the hosting site (e.g., how much network bandwidth it consumes) and remote sites completely unaffiliated with Planet-Lab (e.g., sites that might be probed from a PlanetLab node). Furthermore, both the local administrators of a PlanetLab site and PlanetLab as an organization need to collectively set these policies for a given node.
It must make it easy to audit resource usage, so that actions (rather than just resources) can be accounted to slices after the fact. This concern about how users (or their services) affect the outside world is a novel requirement for PlanetLab, unlike traditional timesharing systems, where the interactions between users and unsuspecting outside entities is inherently rare.
Security was recognized from the start as a critical issue in the design of PlanetLab. However, effectively limiting and auditing legitimate users has turned out to be just as significant an issue as securing the OS to prevent malicious users from hijacking machines. For example, a single PlanetLab user running TCP throughput experiments on U.C. Berkeley nodes managed to consume over half of the available bandwidth on the campus gateway over a span of days. Also, many experiments (e.g., Internet mapping) have triggered IDS mechanisms, resulting in complaints that have caused local administrators to pull the plug on nodes. The Internet has turned out to be unexpectedly sensitive to the kinds of traffic that experimental planetary scale services tend to generate.

3.1.4 EGEE gLite

The EGEE gLite approach is built on top of the Globus middleware, the most successful software to build Grid infrastructures. It is currently exploited by a large number of scientists and, as in the case of the Grid’5000 middleware, some components of its architecture can be re-used in implementing the EDSL, EGEE [4] [5] is a European project that aims to create a large Grid computing infrastructure available to the European and global research communities. EGEE was developed in phases; the current is the third. The project focuses on maintaining and developing the gLite middleware.

The WLCG (Worldwide LHC Computing Grid Project), created to support the computing infrastructure for of the Large Hadron Collider, and the EGEE projects share a large part of their infrastructure and operate it in conjunction. For this reason, we will refer to it as the WLCG/EGEE infrastructure.

The gLite middleware stack combines components developed in various related projects, in particular Condor, Globus, LCG, and VDT, extended by EGEE developed services. The goal of this middleware is to provide the user high-level services for scheduling and running computational jobs, accessing and moving data, and obtaining information on the Grid infrastructure and Grid applications. The gLite Grid services follow a Service Oriented Architecture which facilitates interoperability among Grid services and allow easier compliance with upcoming standard (WS-I, WSRF, etc.). The middleware bases the design of most of its Grid Services on Web Services technologies and describe their interfaces by means of the WSDL. Another of the key design principles is the Service Autonomy, that is, the services should be usable also in a stand-alone manner in order to be exploitable in different contexts.

The gLite services are grouped into 5 logical service groups:

- Security Services: Authentication, Authorization, Delegation, Virtual Organization Membership Server, Sandboxing, Auditing, Dynamic Connectivity Service, Encrypted Storage Service
- Information & Monitoring Services: Basic Information and Monitoring Services, Job Monitoring, Service Discovery, Network Performance Monitoring,
- Data Services: Storage Element, Catalogs (Fireman and Metadata Catalog), Data Movement
- Job Management Services: Accounting, Computing Element, Computing Element Monitor, Workload Management, Logging and Bookkeeping, Job Provenance, Job Provenance Index Server, Job Provenance Primary Storage
- Helper Services: Bandwidth Allocation and Reservation, Agreement Service, Configuration and Instrumentation

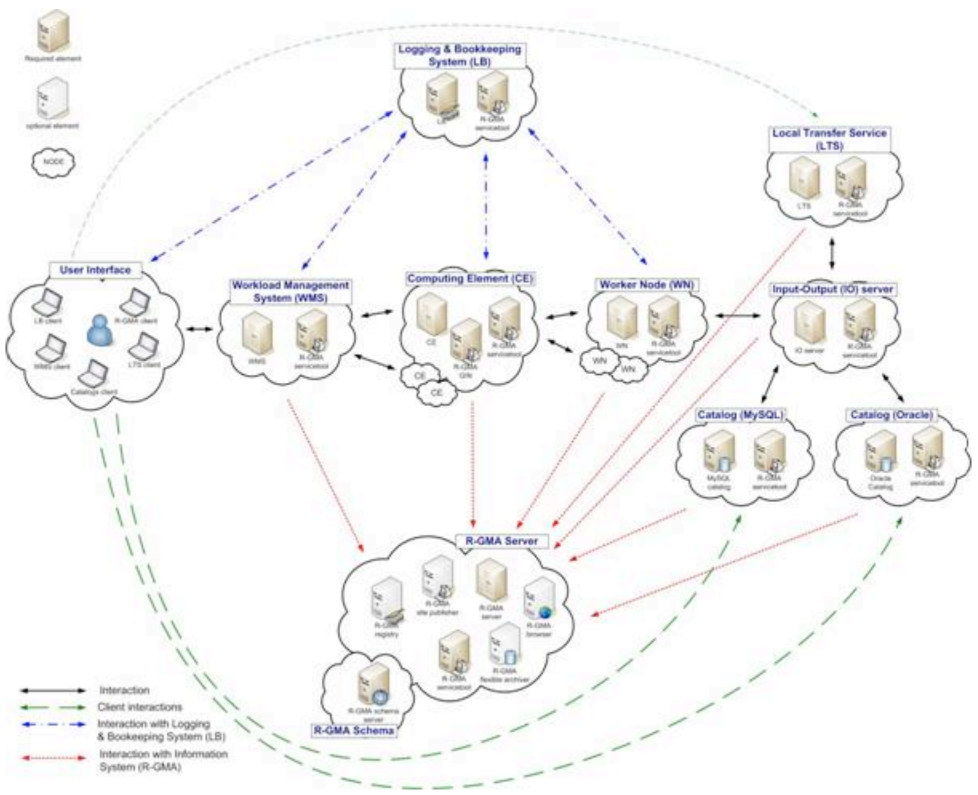

Figure 12: Overview of the gLite architecture [5]

Security

The authentication is ensured by credential storage, containing the main user credentials and providing temporary credentials, while proxy certificates enable single sign-on. TLS, GSI, and WS-Security transport or message-level security protocols ensure integrity, authenticity and (optionally) confidentiality. Attribute authorities enable VO managed access control, while policy assertion services enable the consolidation and central administration of common policy. An authorization framework enables local collection, arbitration, customization and reasoning on policies from different administrative domains, as well as integration with service containers and legacy services. gLite uses myProxy as credential store and the Virtual Organization Membership Service (VOMS) as attribute authority. VOMS is also used to manage the membership of VOs. Most security functionalities are embedded into the service container or in the application itself, for performance reasons. The security architecture should allow a minimal interoperability with other Grid middleware. While accessing to services, the security context is ensured both/either at transport level (SOAP/HTTP/TLS) and at message level (WS-Security). The authentication is verified by means of short-lived Grid credentials provided by a credential store (X.509 certificates) and allows single sign-on and delegation. The authorization is figured out by means of attributes and VO policies obtained in the authentication phase. The service container ensures the enforcement of other security requirements (resource usage, application isolation) at run-time.

Information and Monitoring Services

The gLite system includes a component, the R-GMA, Relational implementation of Grid Monitoring Architecture (GGF specification), for information and monitoring of the infrastructure. Data is written into the R-GMA virtual database by producers and read from it by consumers. The producers have to define the schema, that is, what needs to be published, following the relational model. The consumers submit a SQL query to the virtual database. There are four query types: continuous, latest, history and static, and they are all expressed by a normal SQL query. R-GMA is not a distributed database management system. Instead, it provides a useful and predictable information system built on a much looser coupling of data providers across a grid. R-GMA has been designed to be easy for end users to publish information (from a batch job or otherwise) and query that information in a grid environment.

Workload Management Services

The Workload Management System (WMS) is responsible for the distribution and management of tasks (computational requests, storage or network capacity) over resources, to reach the best efficiency. The core subcomponent, the Workload Manager (WM), accepts and satisfies requests for job management coming from its clients. The decision of which resources should be used is the outcome of a matchmaking process between submission requests and available resources, depending on their state and on the utilization policies (local and VO level). Each job submitted is described in the gLite Job Description Language (JDL), based on the Condor ClassAd language. It has a record-like structure, composed of a finite number of distinct attributes, key-value pairs. A value contains literals and attribute references composed with operators in a C/C++ like syntax. The functionalities the WMS provides are made available through a command line interface and an API providing C++ and Java bindings. GUI components have been developed on top of the Java API. The other subcomponent, the Logging and Bookkeeping service (L&B), gathers information on running jobs. The service can collect system and user information on running jobs. User interface commands and by the public API (simple or more complex queries) make available the job status information.

Data Management Services

In gLite the lowest granularity of the data is on the file level. Each file is identified by the logical file name (LFN). Like a conventional Unix filesystem, the LFN namespace is hierarchical and it has similar semantic. The LFN is not the only name/identifier that is associated with a file; others are:

- the GUID (Global Unique Identifier), a logical identifier;
- the Logical Symlinks, embedding the Unix concept of symbolic links;
- the SURL (Site URL), specifying a physical instance (replica) of a file;
- the TURL (Transport URL), that can be used to transfer a file using any standard transport protocol.

The data management middleware keeps track of logical to physical file instance mappings in a scalable manner by means of catalogs. A catalog, named Logical File Catalogue (LFC), stores the location(s) of their files and replicas. The LFC maps LFNs or GUIDs to SURLs. The data management relies on storage systems exposing a Storage Resource Manager (SRM) interface. Current systems supported include Castor (http://cern.ch/castor), dCache (http://www.dcache.org/) and the gLite Disk Pool Manager (DPM). The DPM has been developed as a lightweight solution for disk storage management offering much of the functionality of dCache but avoiding its complexity. DPM is security enabled, providing ACL based authentication. The current storage systems provide posix-like access libraries, collectively referred as the Grid File Access Library (GFAL).

The gLite File Transfer Service (FTS) is a low level data movement service, responsible for moving sets of files from one site to another while allowing participating sites to control the network resource usage. It is designed for point-to-point movement of physical files. The FTS has dedicated interfaces for managing the network resource and to display statistics of ongoing transfers. There is a set of command line tools available that interact with these interfaces, performing these tasks by contacting the FTS. All the FTS interfaces come with WSDL descriptions and the user can actually use the WSDL to generate clients for any language needed.

3.1.5 Enterprise Service Bus

Today enterprise applications can be seen as complex business processes performed through a huge number of interactions between distinct business partners that have to be integrated across heterogeneous organization domains. In the meantime, Service Oriented Architecture (SOA) offers a flexible and extensible approach to reuse and extend existing applications as well as building new ones, achieving a feasible, scalable and low-cost integration of heterogeneous IT systems. For this reason the SOA approach seems to be the main candidate for designing future enterprise applications. However, to successfully implement an SOA both the applications and the infrastructure on which they are built have to follow the SOA principles: applications' and infrastructure's components must be designed as standardized services that can be reused and assembled to address changing business requirements dynamically. A service can be defined as any discrete and reusable business function that can be offered to an external consumer through an explicit, implementation-independent interface that is invoked using communication protocols that stress location transparency and interoperability. To accomplish an effective and efficient implementation of an SOA within an enterprise it is now emerging the Enterprise Service Bus (ESB): a set of SOA compliant infrastructure capabilities implemented by middleware technologies.

ESB is basically an architectural pattern that acts as an intermediary between service providers and consumers in an SOA: rather than interacting directly, services communicate through the ESB. In this way all the complexity related to services interaction is moved from the communicating partners and encapsulated into the ESB, increasing decoupling between service providers and consumers. This centralization of control through which all interactions are routed is commonly referred to message brokering technologies and described as hub-and-spoke [23] [24]. Despite the fact that the ESB can be seen as a logically single entity that provides a single point of control, it is often described as a distributed infrastructure because it is thought to be deployed as a collection of physically distributed components.

Typically an ESB implementation has to provide a set of minimum capabilities that are often referred to by the acronym "TRANS" [25] which define an ESB as a middleware entity able to:

- **T**ransform messages from one format to another to accommodate the requirements of registered service providers.
- **R**oute messages to registered services while providing quality of service and service level features.
- **A**ugment message content with information such as additional metadata about the message consumer.
- **N**otify registered message listeners about specific message requests.
- **S**ecure delivery of messages by enforcing security mechanisms such as authentication and authorization.

According to a more structured approach [26] [27] [28], following there is a list of state of the art categorized capabilities an ESB should implement: while some are quite fundamental, others represent more advanced autonomic and intelligent features.

- **Communications**
  ESB needs to supply a communication layer to support service interactions through a variety of protocols, provide underlying support for message and event-oriented middleware and integrate with existing HTTP infrastructure and other Enterprise Application Integration (EAI) technologies. The following topics belong to this area: routing, addressing, communication technologies, protocols and standards (e.g. HTTP/HTTPS), publish/subscribe, request/response, fire and forget events, synchronous and asynchronous messaging.

- **Service Interaction**
  ESB has to support SOA principles for the definition of interfaces, declaration of service operations and quality of service requirements. The following topics belong to this area: service interface definition (e.g. WSDL), substitution of service implementation, service messaging models required for communication and integration (e.g. SOAP, XML, or proprietary EAI models), service directory and discovery (e.g. UDDI).

- **Integration**
  ESB should provide linking to a variety of heterogeneous systems that do not directly agree

with service-style interactions. The following topics belong to this area: database (e.g. JDBC), legacy and application adapters (e.g. J2EE™ Connector Architecture), connectivity to EAI middleware, service mapping, protocol transformation, data enrichment, application server environments (J2EE™ and .Net), language interfaces for service invocation (Java™, C/C++/C#).

- **Quality of Service**
  ESB might be required to support service interactions that require different quality of service to protect data integrity among those interactions. The following topics belong to this area: transactions (atomic transactions, compensation, WS-Transaction), various assured delivery paradigms (WS-ReliableMessaging or support for Enterprise Application Integration middleware).

- **Security**
  ESB should keep the integrity and confidentiality of the services that it carries. The following topics belong to this area: authentication, authorization, non-repudiation, confidentiality, security standards (e.g. Kerberos, WS-Security).

- **Service Level**
  ESB should mediate interactions between systems supporting specific performance, availability and other requirements. The following topics belong to this area: performance, throughput, availability, other continuous measures that might form the basis of contracts or agreements.

- **Message Processing**
  ESB needs to be capable of integrating messages, objects and data models between the application components of an SOA. The following topics belong to this area: encoded logic, content-based logic, message and data transformations, message/service aggregation and correlation, validation, intermediaries, object identity mapping, store and forward.

- **Management and Autonomicity**
  ESB needs to have administration capabilities in order to be managed and monitored. In addition to these basic features, it should also provide self-healing and dynamic routing and it should be able to react to events to self-configure, heal and optimize. The following topics belong to this area: administration capability, service provisioning and registration, logging, metering, monitoring, integration to systems management and administration tooling, self-monitoring and self-management.

- **Modeling**
  ESB should support the increasing list of cross-industry and vertical standards in both the XML and Web Services areas. The following topics belong to this area: object modeling, common business object models, data format libraries, public versus private models for business-to-business integration, development and deployment tooling.

- **Infrastructure Intelligence**
  ESB should be able of evolving towards a more autonomic, on-demand infrastructure. The following topics belong to this area: business rules, policy-driven behavior, particularly for service level, security and quality of service capabilities (e.g. WS-Policy), pattern recognition.

Although many ESB implementations leverage on service connectivity standards such as XML and Web Services, the use of these basic technologies on their own is not sufficient to build an ESB. Supporting Web Services is highly desirable within the ESB as is support for other technologies that are required to fully implement an ESB infrastructure. In the following it will be described a specific lightweight open source ESB implementation provided by Apache Software Foundation called ServiceMix.

Apache ServiceMix [29] is an open source distributed ESB built from the ground up on the Java Business Integration (JBI) specification JSR 208. The goal of JBI is to specify a Java standard for structuring business integration systems according to SOA principles. It defines an environment for plug-in components that interact using a service model that interoperate through mediate message exchange based directly on WSDL 2.0 [30] [31].

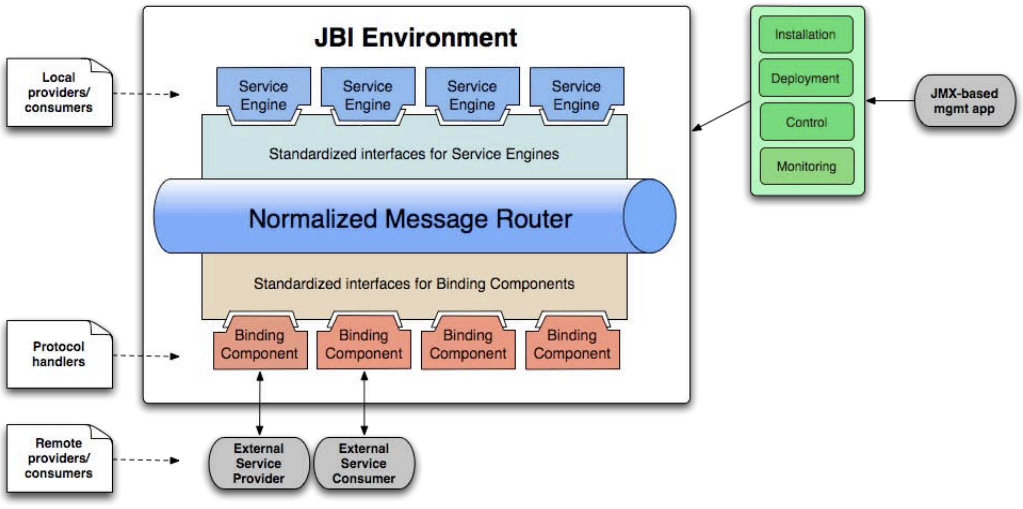
JBI Environment
Local providers/ consumers
Service Engine
Service Engine
Service Engine
Service Engine
Standardized interfaces for Service Engines
Normalized Message Router
Standardized interfaces for Binding Components
Protocol handlers
Binding Component
Binding Component
Binding Component
Binding Component
Remote providers/ consumers
External Service Provider
External Service Consumer
Installation
Deployment
Control
Monitoring
JMX-based mgmt app

Figure 13: JBI Architecture

There are two types of components that can be plugged into the bus: Service Engine and Binding Component. Service Engine (SE) is a component that encapsulates business logic, data transformation, routing and message coordination locally within the JBI environment. It can play both the roles of service consumer or service provider. Binding Component (BC) is used to send and receive messages across heterogeneous communication protocols external to the bus. It decouples the JBI environment from external one by marshalling and unmarshalling messages to and from protocol-specific data formats, allowing the JBI environment to process only normalized messages. As SE, also BC can play both the roles of service consumer or service provider.
Regardless of type (SE or BC) and role (consumer or provider), all plug-in components in the JBI environment communicate in the same way by passing messages using the bus as an intermediary. The JBI Normalized Message Router component is responsible to route the related normalized messages from one component to another. A normalized message consists of three parts:

- message content or payload: an XML document conformed to an abstract WSDL message type;
- message properties or metadata: this extra data can include security information and in general is referred to message context;
- message attachments referenced by the payload and the related handlers.

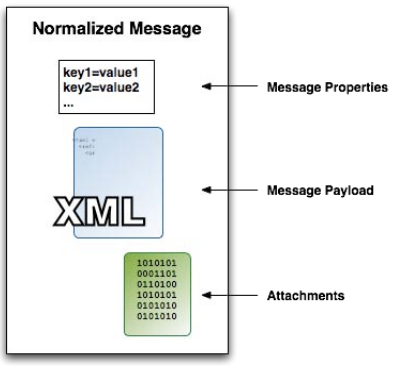

Figure 14: JBI Normalized Message Structure

ServiceMix comes with an ever-increasing array of pre-built components, covering both SEs and BCs.

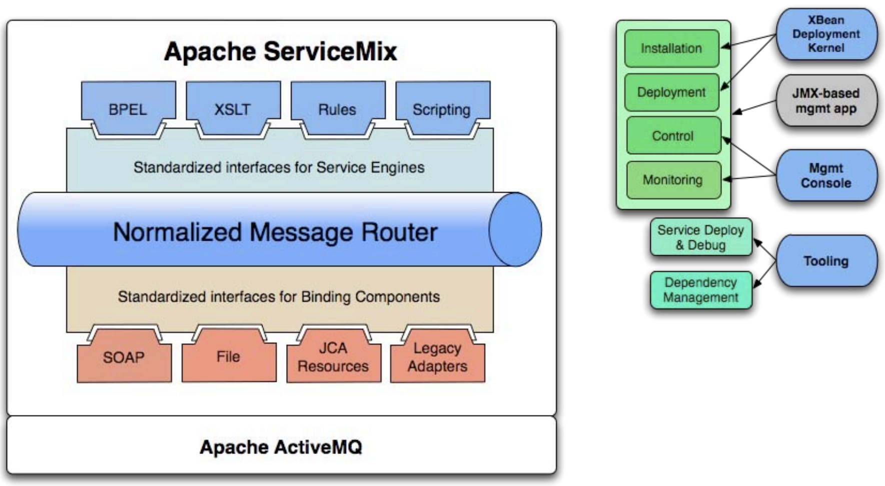

Figure 15: Apache ServiceMix JBI Implementation

The Service Engine and Binding Components available include the ones listed in the two tables below [32].

Engines	**Description**
Quartz	A component that works with the Quartz scheduling engine, allows you to set multiple scheduled service invocation using ServiceMix.
Drools	A component for rules based routing of messages. Flexible XML schema based “domain specific language” can be used to specify rules. Based on Rete algorithm (used in Expert Systems).
BPEL	Business Process Execution Language for Web Services. A standards-based language for the orchestration of components and work flow in business processes.
Groovy	A component for very flexible implementation of services, transformers, and routing logic. The code can be written in the Groovy scripting language and interpreted by the component at runtime.
XSLT based Transformation	A component for transforming incoming normalized messages according to a deployed XSLT template.
Validation	A component providing schema validation of messages.
Generic scripting	A component enabling any JSR 223 compliant scripting language to be plugged into ServiceMix.
XSQL	A component supporting Oracle’s tool for transforming SQL query to XML or persisting XML to database.

Table 1. Built-in Service Engine components in ServiceMix

Binding	Description
Email	A component that allows you to send messages via email, implemented using a Spring framework component and JavaMail.
HTTP	Accepts incoming messages via HTTP. This also includes an HTTP invoker component that can send messages via HTTP.
FTP	Sends messages to an FTP server as files. Also a component to poll an FTP server for incoming file/messages.
JMS	Sends and receives messages via JMS.
JCA	Implements the required support framework of JCA 1.5 (Java Connector Architecture). This component allows compatible connector (typically JMS connector at this time) to run with ServiceMix applications. This component can provide a high performance interface to JMS brokers.
RSS	A component for interfacing with RSS feeds and clients. Can poll feeds and generate messages or transform messages as outgoing RSS.
Jabber	A component for communications via the Jabber P2P network with the XMPP protocol. You can send and receive messages from the Jabber network, and the component even supports the group chat protocol.
VFS	A component implementing access to and from file systems (including OS file systems, jars and zips, Samba (CIFS), etc).
SAAJ	Invokes Web Services using SOAP with Attachments and Apache Axis.
XFire	Supports the XFire lightweight SOAP stack – allowing POJOs to be bound as Web Services.

Table 2. Built-in Binding Components in ServiceMix

Basically there are two development roles within Apache ServiceMix: application developer and container developer. Application developer is focused simply on using the Apache ServiceMix container and exploit the functionalities it provides through its built-in components in order to develop integrated business applications. On the other hand, container developer extends and improves the container capabilities. Application developers spend their time creating so-called JBI Service Units and chaining them together to create a flow of sorts. A Service Unit (SU) is an artifact representing a configuration that instructs a specific JBI component on which services that component has to provide or consume. In order to accomplish this, the SU has to be deployed and plugged into the JBI component. Following an analogy, the JBI component can be seen as a Web container, and SU can be seen as the WAR file that web application developer has to deploy on the Web container.
There are four core elements on which Apache ServiceMix is based, namely the OSGi runtime (Apache Felix), the message broker (Apache ActiveMQ), the SOAP support (Apache CXF) and finally the routing engine (Apache Camel). Through Apache Camel, Apache ServiceMix supports all JBI compliant routing mechanisms, in particular:

1. a component can choose the exact service endpoint on which to invoke operations;
2. a component could act as a proxy in the pipeline, where it can query some registry or repository to find a suitable service with which to interact;

3. a component could use rules (see Drools) or scripting (see Groovy) to decide which service endpoints to invoke;
4. a component can let the container find the service on which to invoke operations;
5. a component can give the container some hints by specifying the service, interface and/or operation to invoke and then let the container choose which physical service endpoint to invoke. The container chooses such physical endpoint using some preferred algorithm (e.g. using rules or policy driven metadata), assuming there may be many service instances available for a specific service, interface and operation names.

Apache ServiceMix also supports also other customizable and pluggable mechanisms. However, currently there is no way for a Web Service provider to expose or register itself on the bus at run-time, even because ServiceMix actually does not support a UDDI service registry. Theoretically a Binding Component could use a UDDI registry to browse for external service providers, and for each one it could register an endpoint reference on the bus: in such a way it will be possible to register external services automatically.
ServiceMix is lightweight and easy embeddable, has integrate Spring support and can be run at the edge of the network (inside both a client or a server), as a standalone ESB provider or as a service inside another ESB. It could depend both on JavaSE or JavaEE application server; currently it is completely integrated into Apache Geronimo, allowing it to deploy directly on it JBI components and services. However, ServiceMix has been also integrated into other JavaEE application servers implementations, like JBoss and JOnAS.

3.2 Collaborative Approaches

Collaborative components are state-of-the-art methodologies, technologies and tools that enable geographically distributed research groups and institutions to collaborate and coordinate their activities. Thanks to the growth of the Web2.0 era, new application development patterns and technologies are increasingly burst into the software market.
We have chosen to describe the following items that fall into this category: Collaborative Software Development Management Systems (CSDMS) such as RCS, CVS, SVN, GForge and SourceForge, and Wiki technologies. We believe that these technologies will have in the short period a major impact in the usefulness of the S-Cube EDSL, allowing the partners to start up quickly collaborations and the sharing of software products and data,

3.2.1 Collaborative Software Development Management Systems

Modern software developers need collaborative technologies and tools to share their work across heterogeneous and geographically distributed environments. Such tools are referred to as Collaborative Software Development Management Systems (CSDMS) and can offer two main functionalities: Source Control and Software Configuration Management. The former maintains a repository of source files, tracking and controlling all changes, while the latter enables source repository management.
Source Control provide the following capabilities:

- sharing: supporting concurrent development;
- versioning: providing version number and dates;
- change tracking: finding change details;
- archival: reproducing any file from any point;
- documentation.

Enabling code sharing through Source Control permits multiple developers to work on the same source base without colliding. In order to avoid collision risks between concurrent developers, two mechanisms can be alternatively provided:

1) locking: locks individual source files so only one person at a time can modify it;
2) merging: allows multiple person to modify a source file and the system will automatically merge the differences.

Locking works fairly well if developers work on different areas of the project and don't conflict too often. However there are two issues with this approach: firstly people who lock files often forget to unlock them when finished. Secondly there is a simple workaround to this problem, just modifying a local copy of the locked file and checking in it when the file is unlocked: in this way it is easy to lose changes. With merging, instead, each developer merges local copies of source files with the latest copy in the database before committing her/his changes: even if this process is normally done automatically by the system, it is a good practice to verify the result of the merging operation before accepting it. Software Configuration Management supports branching through which it is possible to distinguish between development and release version of software.
There are many of tools under the CSDMS umbrella: we discuss the most relevant ones.

Revision Control System (RCS)

Revision Control System (also known as Version Control System (VCS), Source Control or Source Code Management (SCM)) is the management of multiple revisions of the same unit of information and offers only source control capabilities (file management). It is most commonly used in engineering and software development to manage ongoing development of digital documents like application source code, art resources such as blueprints or electronic models, and other projects that may be worked on by a team of people. Changes to these documents are usually identified by incrementing an associated number or letter code, termed the revision number, revision level, or simply revision and associated historically with the person making the change. A simple form of revision control, for example, has the initial issue of a drawing assigned the revision number "1". When the first change is made, the revision number is incremented to "2" and so on. Collision avoidance is provided by a locking mechanism and there is a little support for binary files.

Concurrent Versions System (CVS)

In the world of open-source software, the Concurrent Versions System (CVS) was the tool of choice for version control for many years. CVS was open-source software itself, fitting the collaborative nature of the open-source world very well, and supports both source control and software configuration management. CVS is built on top of RCS but the collision avoidance is provided by a merging mechanism instead of a locking one; it is fast and integrated in an important text editor like Emacs. CVS uses a client-server architecture: a server stores the current version(s) of a project and its history, and clients connect to the server in order to check out a complete copy of the project, work on this copy and then later check in their changes. Typically, the client and server connect over a LAN or over the Internet, but client and server may both run on the same machine if CVS has the task of keeping track of the version history of a project with only local developers. The server software normally runs on Unix (although at least the CVSNT server supports various flavors of Microsoft Windows and Linux), while CVS clients may run on any major operating-system platform. Several developers may work on the same project concurrently, each one editing files within their own "working copy" of the project, and sending (or checking in) their modifications to the server. To avoid the possibility of people stepping on each other's toes, the server will only accept changes made to the most recent version of a file. Developers are therefore expected to keep their working copy up-to-date by incorporating other people's changes on a regular basis. The CVS client, requiring manual intervention only when a conflict arises between a checked-in modification and the yet-unchecked local version of a file, mostly handles this task automatically.

Subversion (SVN)

Subversion has been designed to be a successor to CVS by creating an open-source system with a design (and “look and feel”) similar to CVS, and by attempting to avoid most of CVS's noticeable flaws. Subversion is a free/open-source version control system. That is, Subversion manages files and directories, and the changes made to them, over time. This allows you to recover older versions of your data, or examine the history of how your data changed. In this regard, many people think of a version control system as a sort of “time machine”. Subversion can operate across networks, which allows it to be used by people on different computers. At some level, the ability for various people to modify and manage the same set of data from their respective locations fosters collaboration. Progress can occur more quickly without a single conduit through which all modifications must occur. Because the work is versioned, you need not fear that quality is the trade-off for losing that conduit if some incorrect change is made to the data: just undo that change. Subversion, is not a software configuration management system: it is a general system that can be used to manage any collection of files.

GForge

GForge is an online SCM system that has a set of tools to help developers’ teams to collaborate each other. This set of tools contains message forums and mailing lists and tools to create and control access to Source Code Management repositories like CVS and Subversion. GForge automatically creates a repository and controls access to it depending on the role settings of the project.

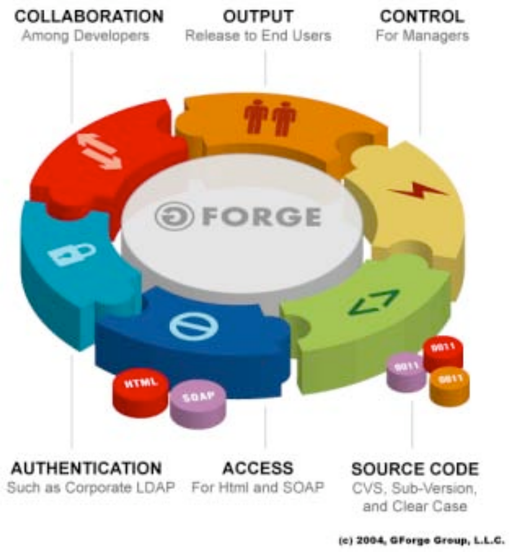

Figure 16: GForge Logical Architecture

SourceForge/SourceForge.net

SourceForge Enterprise Edition is a collaborative revision control and software development management system. It provides a front-end to a range of software development lifecycle services and integrates with a number of free/open source software applications (such as PostgreSQL and Subversion). While originally itself open source software, SourceForge was commercialized as v2.5 prototype code and eventually relicensed under a proprietary software license as SourceForge Enterprise Edition. The original codebase was forked by the GNU Project as Savane. It was also later forked as GForge by one of the SourceForge programmers. Originally sold by VA Software, SourceForge Enterprise Edition was acquired by CollabNet. SourceForge.net is a source code repository and acts as a centralized location for software developers to control and manage open source software development. SourceForge.net is operated by Sourceforge, Inc. (formerly VA Software) and runs a version of the SourceForge software, forked from the last open-source version available. A large number of open source projects are hosted on the site (it had reached 169,281 projects and 1,789,014 registered users as of 2008, although it does contain many dormant or single-user projects). SourceForge.net has offered free access to hosting and tools for developers of free/open source software for several years, and has become well known within such development communities for these services.

3.2.2 Wiki

Wiki is a piece of server software that allows users to freely create and edit Web page content using any Web browser [33] [34]. Wiki supports hyperlinks and has simple text syntax for creating new pages and cross-links between internal pages on the fly. Wikis are typically used as shared whiteboards that allows users to add, remove, or otherwise edit all content very quickly and easily. Like many simple concepts, "open editing" has some profound and subtle effects on Wiki usage. Allowing everyday users to create and edit any page in a Web site is exciting in that it encourages democratic use of the Web and promotes content composition by nontechnical users. The ease of interaction and operation makes a plain wiki an effective tool for collaborative writing and to share knowledge. A structured wiki combines the benefits of the seemingly contradicting worlds of plain wikis and database systems. This provides a collaborative database environment where knowledge can be shared freely, and where structure can be added as needed. In a structured wiki, users can create wiki applications that are very specific to their needs, such as call center status boards, to-do lists, inventory systems, employee handbooks, bug trackers, blog applications and more. Ward Cunningham, and co-author Bo Leuf, in their book "The Wiki Way: Quick Collaboration on the Web" described the essence of the Wiki concept as follows: *A wiki invites all users to edit any page or to create new pages within the wiki Web site, using only a plain-vanilla Web browser without any extra add-ons.*
Wiki promotes meaningful topic associations between different pages by making page link creation almost intuitively easy and showing whether an intended target page exists or not. A wiki is not a carefully crafted site for casual visitors. Instead, it seeks to involve the visitor in an ongoing process of creation and collaboration that constantly changes the Web site landscape. A wiki enables documents to be written collaboratively, in a simple markup language using a Web browser. A single page in a wiki website is referred to as a wiki page, while the entire collection of pages, which are usually well interconnected by hyperlinks, is the wiki. A wiki is essentially a database for creating, browsing, and searching through information.
A defining characteristic of wiki technology is the ease with which pages can be created and updated. Generally, there is no review before modifications are accepted. Many wikis are open to alteration by the general public without requiring them to register user accounts. Sometimes logging in for a session is recommended, to create a "wiki-signature" cookie for signing edits automatically. Many edits, however, can be made in real-time and appear almost instantly online. This can facilitate abuse of the system. Private wiki servers require user authentication to edit pages, and sometimes even to read them. Wiki software is a type of collaborative software that runs a wiki system. This typically allows web pages to be created and edited using a common web browser. It is usually implemented as a software engine that runs on one or more web servers, with the content stored in a file system and changes to the content stored in a relational database management system.
The first such system was created by Ward Cunningham in 1995, but given the relative simplicity of the wiki concept, a large number of implementations now exist, ranging from very simple "hacks" implementing only core functionality to highly sophisticated content management systems. The primary difference between wikis and more complex types of content management systems is that wiki software tends to focus on the content, at the expense of the more powerful control over layout seen in CMS software like Drupal, WebGUI, and Joomla! or at the expense of non-wiki features (news articles, blogs,..) like those in TikiWiki.
Wiki software could be taken as comprising all of the software required to run a wiki, which might include a web server such as Apache, in addition to the Wiki engine itself, which implements the wiki technology. In some cases, such as ProjectForum, or some WikiServers, the web server and wiki engine are bundled together as one self-contained system, which can often make them easier to install.
The majority of engines are open source/free software, often available under the GNU General Public License (GPL); large engines such as TWiki and the Wikipedia engine, MediaWiki, are developed

collaboratively. Many wikis are highly modular, providing APIs that allow programmers to develop new features without requiring them to be familiar with the entire codebase.
It is hard to determine which wiki engines are the most popular, although a list of lead candidates include TWiki, MoinMoin, PmWiki, DokuWiki and MediaWiki. TWiki and Atlassian Confluence are popular on intranets. Following there is an overview of several popular Wiki implementations and see how they fare. It is not trivial to switch from one Wiki implementation to the other, because this will usually require translating all of the pages from the old syntax to the new one. Thus, choosing a Wiki engine requires some care, taking possible future developments into account.

Common features offered by Wiki implementations

For their pages, Wiki implementations offer a markup language, usually very different than HTML. The alternate syntax helps to prevent HTML injection attacks and also allows easier editing. Some Wikis support a subset of HTML, but this is uncommon. Most implementations identify Wiki pages in the markup by using an identifier, usually formed from capitalized words (for example, ThisIsAWikiPage). Wiki users refer to this form of capitalization as CamelCase. Any such identifiers in the page create links to other internal pages in the Wiki. Following a link to a non-existing page often gives the opportunity to edit the page's contents, thus creating a new page.
There are other common rich-text paradigms. Some allow users to designate portions of text as external links to URLs. One embeds images based on the URLs. It's usually possible to highlight text with bold, underline, or italics, add numbered (<ol>s) and bulleted lists (<ul>s), blocks of monospace text (representing program code or output), and so forth.
Some Wikis have page attachments: binary files associated with the page to which it can refer. This is useful for putting images or directly downloadable files in the Wiki. Another common feature is version control. Version control allows users to save and recall previous version of the edited pages, in order to reverse damage, either accidental or malicious. Most Wiki implementations also have a search feature for searching their pages.
In addition to the above, several Wiki engines offer extra features such as user management and permissions, extendibility, and an extended markup that supports various operations such as meta-syntax.

4 Conclusions

This deliverable describes the asymptotic architecture of the S-Cube Pan European Distributed Laboratory. As a first step, we presented the EDSL logical architecture as a layered architecture and then we described the steps we plan to perform to fully implement our overall vision. Behind this vision there is the concept of “island” with some implementation scenarios as examples. Eventually, we presented a detailed review of the state-of-the-art technologies and tools. We believe that such technologies might be successfully evaluated to select the first “building blocks” of the EDSL.
This document has been strongly focused on run-time infrastructures, and there is a lack of description of development/test tools. We plan to include in the next deliverables the description of new selected tools according to the requirements (requests/feedbacks) received by the S-Cube partners. In the next deliverable (CD-IA-1.2.2: Detailed requirements analysis for a Distributed Services Laboratory Network), due at Month 9, both functional requirements for the EDSL and non-functional requirements will be detailed.

5 References

[1] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, O. Richard: Grid'5000: A Large Scale and Highly Reconfigurable

Grid Experimental Testbed (International Conference on Grid Computing, Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, 2005)

[2] Sebastien Varrette, Sebastien Georget, Johan Montagnat, Jean-Louis Roch and Franck Leprevost: Distributed Authentication in GRID5000 (LNCS, On the Move to Meaningful Internet Systems 2005: OTM Workshops)

[3] T. Glatard, J. Montagnat, X. Pennec: An experimental comparison of Grid5000 clusters and the EGEE grid (Workshop on Experimental Grid testbeds for the assessment of large scale distributed applications and tools (EXPGRID'06), 2006)

[4] EGEE gLite User Guide (http://glite.web.cern.ch/glite/documentation/)

[5] E. Laure, S. M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, A. Edlund: “Programming the Grid with gLite” (Computational Methods in Science and Technology, 2006 - man.poznan.pl)

[6] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: “Xen and the Art of Virtualization” (SOSP 2003).

[7] Xen 3.0 interface manual: http://www.cl.cam.ac.uk/research/srg/netos/xen/readmes/interface.pdf

[8] Abels, T., Dhawan, P., Chandrasekaran, B.: “An Overview of Xen Virtualization” (August 2005)

[9] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In Proc. HotNets–I, Princeton, NJ, Oct 2002.

[10] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica.Wide-area cooperative storage with CFS. In Proc. 18thSOSP, pages 202–215, Banff, Alberta, Canada, Oct 2001.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In Proc. 9th ASPLOS, pages 190–201, Cambridge, MA, Nov 2000.

[12] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-Scale Persistent Peer-to-Peer Storage Utility. In Proc. 18th SOSP, pages 188–201, Banff, Alberta, Canada, Oct 2001.

[13] L. Wang, V. Pai, and L. Peterson. The Effectiveness of Request Redirection on CDN Robustness. In Proc. 5th OSDI, pages 345–360, Boston, MA, Dec 2002.

[14] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In Proc. 18th SOSP, pages 131–145, Banff, Alberta, Canada, Oct 2001.

[15] Y. Chu, S. Rao, and H. Zhang. A Case For End System Multicast. In Proc. SIGMETRICS 2000, pages 1–12, Santa Clara, CA, Jun 2000.

[16] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: Offering Internet QoS Using Overlays. In Proc. HotNets–I, Princeton, NJ, Oct 2002.

[17] M. Balazinska, H. Balakrishnan, and D. Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In Proc. Pervasive 2002, pages 195–210, Zurich, Switzerland, Aug 2002.

[18] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In Proc. INFOCOM 2002, pages 1190–1199, New York City, Jun 2002.

[19] A.Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proc. 18th Middleware, pages 329–350, Heidelberg, Germany, Nov 2001.

[20] I.Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Peer-to-Peer Lookup Service for Internet Applications. In Proc. SIGCOMM 2001, pages 149–160, San Diego, CA, Aug 2001.

[21] B. Chun, J. Lee, and H. Weatherspoon. Netbait: a Distributed Worm Detection Service, 2002. http://netbait.planet-lab.org/.

[22] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A facility for distributed internet measurement. In Proc. 4th USITS, Seattle, WA, Mar 2003.

[23] IBM Redbooks: “Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6” (2005) (http://www.redbooks.ibm.com/abstracts/sg246494.html)

[24] Victor Grund, Chuck Rexroad: “Enterprise Service Bus implementation patterns” (December 2007) (http://www.ibm.com/developerworks/websphere/library/techarticles/0712_grund/0712_grund.html)

[25] J. Jeffrey Hanson, JavaWorld.com: “ServiceMix as an enterprise service bus” (December 2005) (http://www.javaworld.com/javaworld/jw-12-2005/jw-1212-esb.html)

[26] Rick Robinson: “Understand Enterprise Service Bus scenario and solutions in Service-Oriented Architecture, Part 1” (June 2004) (http://www-128.ibm.com/developerworks/library/ws-esbscen/)

[27] Rick Robinson: “Understand Enterprise Service Bus scenario and solutions in Service-Oriented

Architecture, Part 2" (June 2004) (http://www.ibm.com/developerworks/webservices/library/ws-esbscen2.html)

[28] Rick Robinson: "Understand Enterprise Service Bus scenario and solutions in Service-Oriented Architecture, Part 3" (June 2004) (http://www.ibm.com/developerworks/webservices/library/ws-esbscen3/)

[29] Apache ServiceMix (http://servicemix.apache.org/home.html)

[30] Ron Ten-Hove, Sun Microsystems: "JBI Components: Part 1 (Theory)" (2006) (https://open-esb.dev.java.net/public/pdf/JBI-Components-Theory.pdf)

[31] Ron Ten-Hove, Sun Microsystems: "Using JBI for Service-Oriented Integration (SOI)" (January 2007) (https://open-esb.dev.java.net/public/whitepapers/JBIforSOI.pdf)

[32] Sing Li: "Plug into JBI with ServiceMix" (November 2005)

(http://www.itarchitect.co.uk/articles/display.asp?id=222)

[33] http://en.wikipedia.org/wiki/Wiki

[34] http://en.wikipedia.org/wiki/Comparison_of_wiki_software