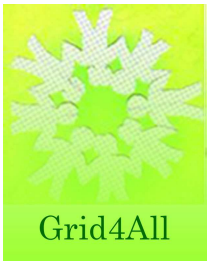

Project no. 034567

Grid4All

Specific Targeted Research Project (STREP)

Thematic Priority 2: Information Society Technologies

D5.1 Evaluation and test plan

Due date of deliverable: 1st June 2007

Actual submission date: 10th September 2007

Start date of project: 1 June 2006 Duration: 30 months

Organisation name of lead contractor for this deliverable: FT

Contributors: Jean-Michel Busca, Nikos Parlavantzas, Gilles Fedak, Andreas Papasalouros, Zenon Perise, Leandro Navarro, Alicia Bou, Eduardo Gomez, Lamia Benmouffok, Ruby Krishnaswamy, Sarfraz Ashfaq, Jean-Michel Busca, Hamid-Reza Mizani, Marc Shapiro, Pierre Sutra, Patrick Valduriez, Vladimir Vlassov, Georgios Tsoukalas, Nectarios Koziris, Aris Sotiropoulos, Per Brand

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Grid4All list of participants

Role	Participant N°	Participant name	Participant short name	Country
CO	1	France Telecom	FT	FR
CR	2	Institut National de Recherche en Informatique en Automatique	INRIA	FR
CR	3	The Royal Institute of technology	KTH	SWE
CR	4	Swedish Institute of Computer Science	SICS	SWE
CR	5	Institute of Communication and Computer Systems	ICCS	GR
CR	6	University of Piraeus Research Center	UPRC	GR
CR	7	Universitat Politècnica de Catalunya	UPC	ES
CR	8	ANTARES Produccion & Distribution S.L.	ANTARES	ES

1. Executive Summary

This document is part of a research project partially funded by the IST programme of the European Commission as project number IST-FP6-034567. This report is a public version of the document "D5.1 – Evaluation and test plan" as specified in the Grid4All Annex 2 "Description of Work". The objective of Grid4All is to provide middleware support and higher level services for the creation and maintenance of virtual organisations formed of autonomous entities within an open environment. The aim of this project is to address non-conventional Grid users, that is, not just large enterprises and scientific institutions, but small organisations, and individual users on the Internet. This report corresponding to the deliverable report D5.1 presents the evaluation plan for the software modules developed within the individual work packages and descriptions of use scenarios against which the developed software will be evaluated taking into account the environmental conditions that are representative of Grids in wide area and open networks.

The WP5 comprises the tasks of integration, test, and evaluation of software modules that are produced within the scope of Grid4All. This work package undertakes the following tasks: definition of evaluation plans for software modules, identify test-beds where the software may be tested and validated, and finally provide evaluation reports presenting the results and conclusions. The per work package evaluation plans validate that developed software components renders the expected functionality and whose non-functional behaviour is correct with respect to a set of identified changes in environment.

The work package 4 provides applications and use scenarios incorporating the environmental conditions that is representative of real-world scenarios targeted by Grid4All. The WP5 uses a subset of these scenarios as yardsticks for evaluation of integrated software stacks.

2. Introduction

WP5 is in charge of integrating the different middleware blocks developed within the work packages WP1-WP4 in an iterative fashion in order to build the Grid4All middleware infrastructure.

Catering to collaborative and service oriented virtual organisations in an environment characterised by scale, heterogeneity, diversity, volatility requires that the underlying middleware and infrastructure addresses issue of autonomy and self-management, that is, endows the system with the ability to reconfigure itself to handle changes in its environment. Grid4All addresses this by combining two paradigms, namely peer-to-peer overlays and component models. Peer-to-peer systems address low level self-management within large area networks, in particular addresses reconfiguration in face of churn and volatility, replication as a response to increase in load etc. Architecture based component models propose a rigorous framework for the design and development of large scale software systems. Endowing architecture based design with feedback control loops has enhanced the autonomic capabilities of large-scale component based software. Grid4All combines these two approaches to build management services for scalable and autonomic virtual organisations. Main issues addressed are that of self-healing and self-configuration through low level services for deployment, detection and handling of node failures. Chapter 3 and section 4.1 present the evaluation plan of self-management within virtual organisations. Our first objective is to demonstrate the self-management behaviours and the ease of adding autonomic behaviours to applications and services and at a second level to quantify the overheads and study scalability.

Support for collaborative applications and scenarios as targeted within Grid4All imply providing pertinent and useful tools for data management. Organisation of storage and management of data that may be modified by many participants in a collaborative environment where disconnection is not an exception is an important work area within Grid4All. The evaluation plan of the data management components has identified pertinent applications such as distributed calendars to qualify the pertinence of the technical approaches. Chapter 5 presents the individual plans to test and evaluate the novel features of the Grid4All data management middleware.

The notion of virtual organisations is core to Grid4All; Grid4All provides automated support for the creation and operation of virtual organisations, involving individuals and different types of physical organisations (such as schools, public councils, small enterprises, families) and the pooling of resources from these potentially diverse origins. Within Grid4All, one characteristic we dote to virtual organisations is that of the ability to dynamically grow or shrink in the number of resources and to acquire resources on-demand as a service. One of the key issues is that of where to find resources to populate such virtual organisations when their demand increases and how to assign resources between multiple contending virtual organisations. The sections 4.2 and 4.3 present the evaluation plans for the components of the resource market place and that of the semantic based information services, which is a key service used by the market place actors.

Chapters 3, 4, 5, and 6 provide the evaluation plans for each of the technical work packages. These plans per work package ensure that software components are validated for correct behaviour before being used in a more complex environment. The behaviour of the components is validated across not only functional correctness but also covers non-functional aspects, but without necessarily integrating it with other Grid4All software modules either as a client or as a server. These per work package evaluation plans nevertheless take into account real environmental conditions that Grid4All purports to address and as has been described within the reports delivered by WP4. This guarantees respect of functional requirements as reported within the requirement analysis and the architecture design of middleware and services (in their respective work packages).

Chapter 7 presents a set of use scenarios, starting from high level applications and going down to the low level infrastructure layer. The scenarios presented within the section will be used as yardsticks to evaluate the Grid4All architecture and middleware. The usability of the middleware and services will be assessed in this phase through experiments and demonstration scenarios that will be derived from these use scenarios to extract results and evaluate the platform. Finally the chapter 8 concludes this first version of evaluation plans and points out the future work in particular as concerns evaluation of integrated software, that needs to be undertaken before the yardsticks set in chapter 7 may be achieved.

3. Evaluation plan for WP1

3.1 Introduction

3.1.1 Objective

The objective of WP1 is to provide infrastructure, tools and models for programming, deploying, and managing Grid services in Grid4All environments. The approach is to base the provided infrastructure on structured overlays (DHTs). The context is virtual organization (VO) that represents a collaborative administrative domain with VO-wide policies regulating membership and the associated obligations and privileges. Applications and services are developed using a component framework, based on Fractal, extended to deal with dynamic virtual organisations whose management framework is built on overlay services.

The Grid4All environments are characterized by the following:

1. High **churn** or dynamicity: In contrast to traditional Grids this means that the identity or availability of constituent parts is continually changing. For example, individual resources can join and leave the VO frequently (or alternatively the availability or quality from the point of view of the VO of a resource varies continuously).
2. **Evolution**: The characteristics of a VO changes in time. VOs grow and shrink as to the resources available to it and the number of members.
3. **Non-professionalism**: It should be possible to create and manage a VO for the purposes of all kinds of collaborations. We cannot assume that they will all have experienced and well-trained system administrators to manage the VO.

A VO can be seen as collaboration involving four different kinds of entities:

1. Resources: Resources are the basis of all collaboration. There are computation resources (computers), storage services (disk space), and bandwidth resources. The goal of a VO is, of course, to make good use of the available resources, in conjunction with VO-policies.
2. Members: The VO is composed of members (these might, in the general case, be VOs in their own right). Associated with members are rights and obligations.
3. Services/applications: The VO provides services and applications that are made available to VO-members.
4. Components: The constituent parts of applications and services. This includes software components and storage. The VO manages an application/service. The application/service is composed of a set of connected components. The components in turn run on (or make use of) resources.

In WP1 we aim to provide the infrastructure through which a VO can be well-managed. From the viewpoint of WP1 the VO management in WP2 is an application making use of the infrastructure primitives and component framework. One particular management framework that we focus on is that of deployment of application level components on the resources forming the virtual organisations. The low level services provided by WP1 are then used by higher level VO management services.

3.1.2 Software packages

We will develop the following software packages in WP1. They are all part of the infrastructure services and the three sensing/actuation services make use of Niche internally. The first three provide an API for the higher-level VO management services. Niche is an extended DKS (a DHT).

Component	Interface technology	Software packaging	External software
Niche	Java API	.jar files	DKS middleware
Resource Sensing and Actuation Service	Java API	.jar files	Niche middleware
Component Management Sensing and Actuation Service	Java API	.jar files	Niche middleware
Membership Sensing and Actuation Service	Java API	.jar files	Niche middleware

Component	API published	Software available	How/where
Niche	2007/07	2007/07	http://korsakov.sics.se/svn/dks/trunk/Niche/

Note that the 3 sensor and actuation services are used directly by the VO management services of WP2.1 (VOMS). They are not intended for direct use (though sometimes the VOMS may make them available with some minimal wrapping.

3.1.3 Evaluation Dimensions

The infrastructure, tools and methods developed in WP1 need to be evaluated along the following dimensions:

1. Functionality: useful & well-defined infrastructure services
2. Non-functionality: efficient well-defined infrastructure services
3. Self-management of basic infrastructure services.
4. Support for building self-managing higher level services, applications and services.

Dimension 1 (functional) : Clearly the infrastructure must provide services that are functionally useful and clear**,** with well-defined semantics, for the higher levels. Higher levels include higher level management, computation, storage services (WP2 and WP3) as well as application (WP4). This is dimension 1.

Dimension 2 (non-functional) : The non-functional properties of the infrastructure (e.g. robustness, performance, scalability) must also be clear.

Dimension 3 (self-managing): The infrastructure services need to be almost completely self-managing. This is because the VO management and developing self-managing applications is challenging in their own right, and the complexity of this would be overwhelming if they also had to deal with partial failure in the infrastructure.

Dimension 4 (self-management support): The infrastructure services should provide good support for higher-level services to build self-managing services, applications and the VO. The division of labour between the infrastructure and the higher levels need to be both clear and natural.

3.1.4 Organization of this section

In the following subsections we describe in more detail the evaluation plan along the aforementioned 4 dimensions. Here we consider the work package as a whole without task distinction.

Thereafter in subsection 1.6 we consider, where necessary, the evaluation from the subtask point-of-view and discuss some task specific evaluation steps.

3.2 Functionality

The infrastructure services present themselves to the higher level services (in particular VO management) in the form of an infrastructure API. This evaluation must be done in conjunction with the other technical work packages (WP2, 3, 4). We need to consider

1. Clarity: The services are semantically well-defined. As in all middleware the complexity of interactions should be kept as low as possible.
2. Usefulness. The provided services are useful to VOMS (developed in WP2.1).

It is, of course, difficult to be objective here, as is the case for all middleware. Clarity and usefulness are somewhat subjective. As the use of overlays in the infrastructure is mostly motivated by the non-functional properties, the evaluation here can be conservative. The point is not to show that functionally this is the most sophisticated possible, but that it either provides sufficient services or that they could be plugged in.

3.3 Non-functional

The infrastructure services need to be evaluated in terms of non-functional properties in the face of dynamic VOs (churn and evolution). This can be divided into:

1. Churn: That the provided services work with good levels of QoS with high rates of churn (particularly as regards resource churn).
2. Evolution. That the provided services scale well (both up and down).

As regards churn and to some extent evolution the services can be evaluated quantitatively. We can plot the various aspects of QoS of given infrastructure services against increasing rates of churn. Hopefully the system can tolerate high rates of churn (and this may be compared to other systems). Also hopefully when churn increases that the infrastructure can both inform management that the limits are being approached before breaking. Graceful degradation is another desirable feature.

3.3.1 Quantitative evaluation

One evaluation, internal to WP l is to provide an evaluation suite testing at least 6 smaller compositions (i.e. a script making use of several infrastructure calls to perform a composite action). These will be plotted against churn and evolution.

For a smaller composition (representing a smaller traditional distributed application) we have:

1. Initial deployment: Finding and allocating resources and deploying a small distributed application.
2. self-configuration: Dynamic reconfiguration of an application in the face of resource availability change

3. self-healing: Recovery from partial failure (i.e. one component fails as the machine fails)
4. self-tuning: Dynamic reconfiguration in the face of load change

Other compositions will reflect test applications making use of the extensions to the component model (largely as regards bindings in the component model to reflect different communication primitives):

5. self-configuration:
6. self-turning:

The qualities to be tested are

1. timeliness (adaptation time)
2. correctness

Churn and evolution are

1. Churn: Resources join and leave (components can be moved to other suitable resource).
2. Churn: Resources fail (crash-stop)
3. Evolution: Growing/shrinking VO
4. Evolution: Load (reflecting popularity of the application/service)

The cross-product of the above is 6x2x4=48 tests (though some are less interesting than others), that can be plotted with quality of service (time or correctness) versus rates of change. Good results would preserve quality even in the face of high rates of churn and evolution

3.3.2 Scale

One of the motivations for basing the infrastructure on structured overlays is that overlays also deal well with scale (the total number of resources and other entities in the system). It is possible that the evaluation of the infrastructure as regards scale can only be supported by simulation.

3.4 Self-management of infrastructure services

This dimension is also related to the non-functional properties of the services as described in the previous subsections. The goal is that the infrastructure services almost completely manage themselves in the face of churn and evolution.

Clearly, there are levels of churn that can break any system, so there needs to be a clear API that lets the higher level management know that the system is close to collapse. Also infrastructure aggregation services need to provide some statistical view of the VO-wide system.

Note that the self-management of the infrastructure services is indirectly being tested by the tests of section 3.3.1 (as breakdown of self-management of infrastructure services will break feedback loops that depend on them).

3.5 Self-management support

A crucial aspect of the infrastructure services is that in conjunction with the VO-management being developed in WP2 that it is possible and relatively easy to build self-managing applications/services. A self-managing application or service in a Grid4All VO is achieved by the cooperation of three different systems/programs.

1. The infrastructure
2. Higher level tools including VO management tools and services developed within WP2
3. The application management logic

The application developer in addition to developing his program, using the component framework of WP1, needs to write programs for managing his application, for deployment and configuration. The developer also needs to specify the actions to be taken in the face of failures in some of the application's components and also when new resources become available within the virtual organisation. The detection of these two events are part of the self-management support provided within WP1.

3.6 Task-specific evaluation

3.6.1 Task 1.1

This is the main task of this work package and the evaluation plan has been covered in the previous subsections.

The output of this task is also validated through the VO management services described in the section 4.2, in particular the resource discovery service and the deployment service.

3.6.2 Task 1.2

In this task a component framework, based on the Fractal component model, is developed for dynamic VOs. Functionally, this component framework, compares to previous mainly cluster-based work, is modified to take into account the Grid4All environment.

The framework needs to reflect the properties of the system to enable application developers to develop applications that work well in this environment. This includes programming (e.g. in ADLs that work with the component model) suitable self-management. In a distributed heterogeneous environment statistical reliability and locality of resources become important. In addition the model of component binding needs to be rich enough to reflect the basic communication mechanisms offered by infrastructure (e.g. communication by name rather than by address, anycast, and multi-cast).

Conceptually, all useful patterns of deployment and adaptation in the face of churn or evaluation must be expressible in the component framework and associated management tools. (This is partly in conjunction with task 2.1. VO management). It should also be relatively simple to use.

This is difficult to evaluate objectively, but should be supported by applications and scenarios making good use of the component framework. Supporting are cases where applications making use of the component framework and the associated management logic perform well on dynamic VOs and where it would be difficult or impossible to achieve the same with non-modified Fractal or other component framework. Note that test 5 and 6 in section 3.3.1 indirectly test the some of the Fractal extensions (as well as the infrastructure).

In addition the extended Fractal model should capture all of the relevant properties of dynamic VOs to allow for developing maximally efficient self-managing applications. The abstractions offered or implied by the component model (or infrastructure) should not preclude solutions that would achieve the same result at a lower level with significantly lower overhead (in terms of messages, or latency delays).

3.6.3 Task 1.3

The work in this task largely feeds into task 1.1. and 1.2. with additional requirements on the overlay services (the infrastructure) and the component model. Evaluation as regards this task is in conjunction with composition of systems (or large subsystems).

3.6.4 Task 1.4

This task is the theoretic framework underlying the work done in WP3 on the semantic store. Evaluation is therefore done in WP3, where implementation based on this model are developed and evaluated.

4. Evaluation plan for WP2

This chapter presents the evaluation plan for each task within the work package.

4.1 Autonomic VO management -- T2.1

This task provides a set of core services for constructing and managing Virtual Organisations (VOs). Moreover, it provides a framework for building autonomic managers that control various VO aspects. The main tested and validated components of this task are described in the following table:

Component	**Description**
VO management services (VOMS)	Offers services for managing VOs; the services support setting-up and dissolving VO, managing membership, deploying application components, discovering resources, allocating resources, monitoring and controlling resource usage of applications.
Autonomic manager framework (AMF)	Supports building autonomic managers that realise feedback loops. Specifically, it provides reusable design and code for composing the main elements of control loops: monitoring, analysis, planning, and executing functionality

The following table provides a summary of the interfacing technology and the software packaging of the two main sets of software modules.

Component	**Interface technology**	**Software packaging**	**External software (non grid4all) used,** **Protocols/specifications used.**	**Development and build tools used**
VOMS	Fractal component interfaces defined in Java	Fractal components packaged as OSGI bundles	Oscar OSGI framework (BSD licence), BeanShell scripting for Java (LGPL), Fractal API, Julia, Fractal RMI, Fractal ADL (LGPL)	
AMF	Same as above	Same as above	Same as above	

Availability

Component	**API publish date**	**Software available**	**How/Where**
VOMS	Deployment service and basic resource discovery service APIs are currently available. Remaining APIs available December 2007.	Deployment service and basic resource discovery service implementations are available now. The next version for evaluation will be available in May 2008.	http://gforge.inria.fr/projects/grid4all/

Component	API publish date	Software available	How/Where
Autonomic manager framework	Design guidance in the form of the "architecture-based management" approach, and associated implementation support (e.g., Fractal interfaces, ADL, controllers) are currently available. Remaining APIs available December 2007.	The implementation currently provides the frameworks for monitoring and deployment. The first prototype of complete framework will be available by May 2008.	http://gforge.inria.fr/projects/grid4all/

Objective of evaluation

Component	Objective of evaluation
VOMS	– Functional correctness of services – Resilience of services – Ease of use of services – Scalability in the dimensions listed below: o Number and geographic distance of nodes o Number of members o Number of deployed application components
AMF	– Functional correctness of basic set of self-management behaviours: o Basic self-healing by relocating component when its underlying node fails o Basic self-configuration by relocating component when a 'better' resource joins the system o Basic self-optimisation by adjusting the number of replicas of a component in order to maintain the average replica load within a specified range – Ease of extension with respect to autonomic managers – Generality of framework – Efficiency of framework – Scalability in terms of number and geographic distance of nodes.

Component	Qualitative and quantitative metrics of satisfaction
VOMS	– Functional correctness o Satisfaction of test-cases for each service; the test cases are derived from use scenarios – Resilience o Experiments in which system continues to support services even when

Component	Qualitative and quantitative metrics of satisfaction
	nodes fail. Measure number of simultaneous failures and failure rates that are handled and compare with failure characteristics of real Internet nodes. – . Ease of use o Complexity of necessary code/ADL descriptions for using services (within given scenarios). – Scalability o Performance of VO management services (response time and throughput) as a function of the scalability dimensions seen previously
AMF	– Functional correctness o Satisfaction of test-cases for the basic self-management behaviours – Ease of extension o Examples of developing and integrating new autonomic managers o Measure lines of code for new managers – Generality of framework o Demonstrate a range of autonomic managers supporting different self-managing properties. Chosen properties are self-healing, self-optimization, and self-configuration. – Efficiency o Quantitative evaluation of framework overhead: running an application with and without self-management support and comparing its performance – Scalability Experiments that show effective self-management of VOs involving many widely-distributed nodes (at least 200 nodes)

Environment assumptions

Component	Test bed	Test data	Environment conditions
VOMS	Cluster with up to 24 machines composed of Intel Core Duo 1,66 GHz/2GB running Linux. Connection through Gigabit Ethernet. Experiments will run on Grid5000 subsets with (at least) 200 processors distributed on (at least) 3 different sites.	To be determined later.	Volatile CPU and network conditions (e.g., variable CPU load, network latencies, bandwidth), heterogeneous nodes (e.g., different CPU speeds, memory/disk capacities), dynamically varying number of VO members and nodes
AMF	Same as above	Same as above	Same as above

Expected functionality and test clients

Component	**Client (target user of functionality)**	**Test method**	**Expected functionality (from Grid4All)**
VOMS	VO manager	Test clients based on test cases for each service	WP1 overlay infrastructure
AMF	Developer of autonomic managers	Develop autonomic managers (e.g., for self-configuration)	WP1 overlay infrastructure

4.2 Market based resource allocation -- T2.2

This task provides a set of components and software modules that enables operation of a Grid resource market place. The main functional components of the Grid4All market place are described in the following table:

Component	**Description**
Market Information Service (MIS)	This service allows participants in the market to provide and obtain market related information by offering interfaces to publish and subscribe. Publishers can post quasi-static or dynamic information. Subscribers can perform queries on index (one-shot-pull) or request notification (subscription). Subscribers may query or request notifications that aggregate the data published thus reducing the cost of distribution. Summary and aggregated information may also be extracted from the MIS.
Currency Management Service (CMS)	This is a P2P based currency management system which will serve as a medium of exchange between all trading agents (buyers and sellers). Account holders may withdraw and deposit currency and perform payments to other account holders.
Market auction server and framework	Tool-kit and framework for development of configurable auction servers. The framework is validated through the implementation of two auction mechanisms (a) a double auction market to trade single type of resources and (b) a combinatorial exchange auction to trade bundles of multiple types of resources.
Market factory service	Software repository/factory of market implementations that allow actors in the market place to select and deploy specific market mechanisms. This factory service offers interfaces to three different types of users: (a) market place administrators (b) developers (c) buyers and seller agents
Agreement Manager	This component establishes the agreement between the winning pairs of consumers and providers. Agreements are established once markets have selected winning buyers and sellers.

Component	Interface technology	Software packaging	External software (non grid4all) used, Protocols/specifications used.	Development and build tools used
MIS	Web Services and Java API	.jar files, .war, and WSDL		Ant based project management tool
CMS	Web Services and Java API	.jar files, .war, and WSDL		idem
Market auction server and framework	Web Services and Java API of Fractal interfaces	.jar files, .war and WSDL	Apache Axis, Optimization software (CPLEX), Fractal Julia	
Market factory Agreement Manager	Web Services and Java API of Fractal interfaces Java API	.jar files, .war and WSDL .jar files	BPEL4WS, WS-CDL. P4SOA, ActiveBPEL To complete end 2007	

The following table describes when APIs and software will be ready for evaluation. Testing and evaluation will be conducted in two steps following progress in maturity and completeness of implementation.

Component	API publish date	Software available	How/Where
MIS	01/10/07	December 2007 June 2008 November 08	http://gforge.inria.fr/projects/grid4all/
CMS	01/09/07	December 2007 June 2008 November 2008	idem
Market auction server and framework	01/10/07	December 2007 June 2008	idem
Market factory Agreement Manager	01/12/07 01/10/07	June 2008 November 2008 December 2007 June 2008	Idem idem

4.2.1 Evaluation

This section presents the validation and evaluation objectives at the level of each functional component. The objectives of the functional component level evaluation are:

- ✓ To verify the correctness of the implemented sources before integration within a more complex use case.
- ✓ To understand the economic performance of different type of market mechanisms.
- ✓ To derive concrete indicators that may be used to suggest correct configurations of systems within a demonstrator that incorporates real-world assumptions on scale.
- ✓ Finally to conclude on the flexibility of the frameworks to enhance with implementations of new mechanisms (where pertinent).

Component	**Objective of evaluation**
MIS	- Overhead of MIS: in global load, number of messages - Scalability in terms of participants, volume of events, degree of aggregation - Self-* behaviour
CMS	- Overhead of CM: in global load, number of messages - Scalability in terms of participants, volume of transactions, effect of concurrency - Self-* behaviour
Market auction server and framework	– Ease of use with respect to development of new auction mechanisms/protocols – Ease of use with respect to configuration and assembly – Auction mechanism validation (for each of the two mechanisms, k-DA and CA) o Scalability (in numbers of buyers, sellers, and bids, size of bids) o Economic and computational performance (efficiency of allocation, fairness) – Overhead of framework
Market factory	*N/A.*
Agreement Manager	– Functional correctness

Component	**Qualitative and quantitative metrics**
MIS	N/A (for 01/12/07)
CMS	- Measure of scalability in transactions (payments) per unit of time. - Measure of errors in respect to percentage of node failures, size of the deployed banking system, and replication factor. - Estimate resource needs for deployment based on number of participants, expected performance and concurrency of transactions (payments).
Market auction server and framework	– Measure of flexibility of framework in the implementation of new auction mechanisms and measure of ease of configuration and assembly of components, based on return of experience from implementations of two auction mechanisms

Component	Qualitative and quantitative metrics
	based on o Lines of new code o Reuse of components – Qualify the mechanism in terms of: o Efficiency of allocation o Fragmentation of resources o Equity, or number of rejected jobs whose values are greater than minimum value of accepted jobs – Derive acceptable (in terms of time to clear auction) upper bounds for both auctions for different configurations of test machines in terms of number of offers, requests, and number of bundles in the offer and requests.
Market factory	N/A: will be produced for 01/02/08
Agreement Manager	N/A: will be produced for 01/11/07

Component	Test bed	Test data	Environment conditions
MIS	Cluster with 50 nodes, PlanetLab Europe, Grid5000 during integrated evaluation	Appropriate data sets representing the publication of information and a collection of queries and subscriptions with different clauses.	Fixed or variable load on nodes, variable number of nodes, injection of failures.
CMS	idem	Appropriate data sets representing payment operations among diverse account holders.	idem
Market auction server	Network of workstations, upto to 12 nodes. Clusters upto 50 nodes, Grid5000 during integrated evaluation	Appropriate data sets representing bids for the k-double auction (single item and multiple units) and for the combinatorial auction (multiple items in bundles). Generated data sets that are representative of the two main scenarios (applications) will be used to test the auction mechanisms and their performance (computational, economic).	Multiple simultaneously executing auction markets where test clients will choose one and participate within. Test clients will represent two categories of applications (a) clients requiring variable quantities of only CPU resources (b) clients requiring variable quantities of CPU and storage bundles
Market factory	idem	Expected 01/02/08	01/02/08
Agreement Manager	Cluster with 50 nodes, PlanetLab Europe, Grid5000 during integrated evaluation	Expected 01/11/07	Different conditions in the type of agreement in involved participants (resource providers and consumers).

The following table describes the test methods that will be used, and the targeted users (clients) of each of the components. Individual test of component will develop test clients which are representative of the real clients. The expected functionality represents the other Grid4All software on which each component is dependent for a complete integrated evaluation (not needed in the scope of the local evaluation).

Component	**Client (user of functionality)**	**Test method**	**Expected functionality (from Grid4All)**
MIS	The expected real clients of service are: auction market services as publishers, trading agents, that is, buyers and sellers of resource as subscribers, and administrators for deployment and configuration.	Client programs written in Java (for both unitary tests and load tests).	Overlay services (WP1) and VO management framework (T2.1) for the deployment.
CMS	Real clients of the service are resource consumers and providers in the market place and the agreement manager that mediates.	Client programs written in Java (for both unitary tests and load tests).	Overlay services (WP1) and VO management framework (T2.1) for the deployment.
Market auction server and framework	Developer (adaptability and reuse of framework). Consumers, providers of resources. In Grid4All the consumers are virtual organisations that allocate resources on demand to satisfy internal computational and storage requirements.	Return of experience from prototyping of two auction mechanisms. Computational and Economic performance of the two selected auction mechanisms through minimal buyer and seller agents representing two different virtual organisations whose scenarios are described in D4.1 and D4.2.	Overlay services (WP1) and VO management framework (T2.1) for the deployment.
Market factory	N/A	N/A	N/A
Agreement Manager	Market auction service that establish transactions between buyers and sellers.	Client programs written in Java (for functional testing).	Overlay Services (WP1)

The Grid4All resource market place addresses assignment and allocation of resources for a wide range of emerging use scenarios within wide area network environments such as the Internet. These dynamic and evolving environments are characterised by:

- Heterogeneity of resources providers in terms of type of resources, size of resources offers, time availability of resources etc.,
- Heterogeneity of applications,

- Diversity in resource requests in all aspects; size of requests, value or priority of requests, characterisation of arrival of requests etc.,
- Change in parameters such as number of resource providers and consumers,
- Incapacity to assume global and complete knowledge of entire system at a given time.

The first evaluation plan described here focuses mainly on validation of functional correctness and flexibility of the component based approach used in the implementation of the main software modules. The next steps in evaluation will focus on the following aspects and provide technical metrics to evaluate the system.

- The end to end integrated testing and evaluation technical step consist of validation and evaluation of a complete use scenario for market based resource allocation, starting from where a virtual organisation detects requirements for new resources and terminating when the resources are released by the virtual organisation at the end of the leased period.
- Deployment of market place components as an overlay service incorporated within the VO management framework consists of integrating the market place components within the VO management framework. The market place will be operated on a peer-to-peer based overlay where deployment of the market place components will themselves be performed by the software modules developed within T2.1.
- Dynamics of system within a real-world scenario where there are multiple competing consumers (virtual organisations) and multiple resource providers.
- Evaluation of the economic and societal impact of the Grid resource market place.

4.3 Semantic information services -- T2.3

The Semantic Information System (SIS) provides a matching service between peers to offer or use resources and services within grid environments. In the market place, resource consumers and providers negotiate traded entities in auction based markets, where these markets are spontaneously initiated (instantiated) by different actors in the market place such as resource providers, consumers, or 3rd party actors. Such instantiated markets are accessed as services which are themselves advertised at the Semantic Information System which acts as a registry. The SIS may be queried by different software agents as well as human users to select advertised markets (through different selection criteria such as resource quantities and quality attributes, prices etc.). The returned query results are ranked according to capacity of resources and preferences.

The main tested and validated components of this task are described in the following table:

Component	**Description**
Matching component	This component matches requests and offers. The output of this component will be a ranked list of markets that requested/offered resources or services can be traded. This component will be used by trading agents such as buyers and sellers. The component will also be used by the portal interface component.
Portal Interface component	This component is used to populate the SIS registry with requests and offers. This component will also provide a web interface for the matching component in order to perform queries. Requests and Offers are internally stored in the SIS as ontology instance data (RDF triples). Data is inserted in the SIS in two ways: a) Through a web-based user interface through which offers and requests are registered. The objective is to render the ontology description transparent to the user. Requests and offers inserted through the web forms are inserted in the underlying storage provided by the SIS. b) Through a web service interface accessible through SOAP. The interfacing component serves the purpose of inserting data into the system both through the user interface by humans and the SOAP interface by software agents.

Component	Description
	The portal component interacts with the Market Information Service (MIS) through a web services interface. This component will be used by human users and software agents, both buyers and sellers.
Selection service	This service reduces the set of relevant providers, found by the Matchmaking Service, by considering the consumers' and providers' interests to allocate and perform queries, respectively. Unlike related work on query allocation that strives to find interesting providers for consumers, we also strive to find interesting queries for providers. With providers we mean those sites that can answer or correspond to the consumers' queries. For example, if a consumer is querying for a given music file (or for markets), the providers are those sites that have such a file (respectively the running markets themselves). In fact, we plan to evaluate if consumers and providers are satisfied with the results and queries they get from the system. This component will be used by the Matching component.

Component	Interface technology	Software packaging	External software (non grid4all) used, Protocols/specifications used.	Development and build tools used
Matching	Java API and WSDL	Web archive (WAR file) deployable within a Tomcat servlet container	J2SE (Sun licence) Jena Ontology API (HP) Pellet (LGPL) or Racer OWL reasoners Jakarta AXIS (apache licence BSD?) Jakarta Tomcat (apache licence BSD?)	
Portal Interface	idem	idem	idem	
Selection service	idem	idem	N/A	

Availability

Component	API publish date	Software available	How/Where
Matching	Java API in December 2007	April 2008	N/A
Portal Interface	Idem	Idem	N/A
Selection service	Idem	Idem	N/A

Objective of evaluation

Component	**Objective of evaluation**
Matching	– Correctness o The percentage of offers that were correctly matched, that is, the number of all offers that were returned by the system as applicable to a particular request divided by the overall offers that are actually applicable to the particular request, given that the system is populated with a set of offers and requests o The percentage of offers that were incorrectly matched, that is, the number of requests that are returned by the system while they are actually not applicable to the particular request divided by the overall requests that were returned by the system. o Measures of precision and recall from information retrieval, as well as harmonic means or F1 measures shall be used. Peer satisfaction measures will also be used. – Performance o The dependence of the performance of matchmaking from the number of available requests and offers will be tested)
Portal	– Ease of use: Human users of the system should be able to fill in advertisements and perform queries without needing knowledge of ontology. The average time to fill in forms will be measured in order to evaluate ease of use. – Correctness: o The correct insertion of data through the SOAP interface o The correct population of data inserted in web forms provided by the portal component o Percentage of successful data insertions will be measured to assess this correctness. – Performance
Selection	– Correctness, that is, validate that the selected providers correspond to the consumers and providers expectations. This validation will be done by analyzing the satisfaction of consumers and providers. We also define the satisfaction notion as a qualitative metric. – Adaptability to high work loads

Qualitative and quantitative metrics

Component	**Qualitative and quantitative metrics of satisfaction**
Matching	– Concrete metrics will be developed in the future
Portal Interface	– Concrete measures will be developed in the future
Selection	– Satisfaction and correctness o Satisfaction of returned results will be measured according to the principles described in *" Jorge-Arnulfo Quiané-Ruiz, Philippe Lamarre, and Patrick Valduriez. SQLB: A Query Allocation Framework for Autonomous Consumers and Providers. In the Procs. of the VLDB*

Component	Qualitative and quantitative metrics of satisfaction
	Conference, 2007 " – Scalability o Increase of workload from 20% to 120% of system capacity, that is, the aggregate capacity (in terms of computational resources) of all the registered providers in the matching and selection service.

Environment assumptions

Component	Test bed	Test data	Environment conditions
Matching	A computer with JDK 1.5 Jena and Racer.	Test data will be based on the use cases specified in D2.1 and scenarios derived from WP4;	A number of 10 queries will be simultaneously performed. Insertions and deletions are allowed during queries. Data integrity is ensured during concurrent data access using appropriate transaction management techniques.
Portal Interface	A computer with JDK 1.5 and Tomcat.		
Selection	A computer equipped with SimJava and JDK1.5	Query arrival modelled using the Poisson distribution. Each test will be repeated 10 times with: – 200 consumers – 400 providers – 1 mediator	

Expected functionality and test clients

Component	Client (target user of functionality)	Test method	Expected functionality (from Grid4All)
Matching	Unitary test clients linked to the matching component	Unit tests using the JUnit framework	None
Portal Component	Clients using Cactus or HttpUnit frameworks		None
Selection	SimJava based simulator		None

4.4 Scheduling service -- T2.4

The Scheduling Service provides a matching service between applications, presented as a set of tasks and hosts, presented as a finished set of nodes. The result of this matching, and therefore the output of the scheduling service is an execution plan presented as a Gantt graph which indicates on which machines and at what time should tasks be executed. This service will work in cooperation with the Market Based resource Management System (GRIMP) provided by task 2.2, for resource acquisition, and the deployment service provided by the task 2.1 for deployment of tasks onto resources and their life cycle management.

The main tested and validated components of this task are described in the following table:

Component	**Description**
Scheduling component	This component plans the schedule of application execution on compute nodes. It will implement classical scheduling heuristics for bags of task class of applications. This component will not be used directly by other components but through the interface component.
Interface component	This component is used by other components to obtain an execution plan. The result of the component is a Gantt chart. This plan may be used either to provide an actual deployment and execution plan, ready to be used by the deployment service or can be used as an execution simulation. For example, this simulation of execution may be used by the buyer agents acquiring resources from the resource market to estimate an execution time and make decision about resource acquisition.

Component	**Interface technology**	**Software packaging**	**External software (non grid4all) used,** **Protocols/specifications used.**	**Development and build tools used**
Scheduling	Java API	Java archive (Jar file)	GPL licence	Ant
Interface	idem	idem	GPL and LGPL to wrap the GPL code	

Availability

Component	**API publish date**	**Software available**	**How/Where**
Scheduling	Java API in February 2008	November 2008	N/A
Interface	Idem	June 2008	N/A

Objective of evaluation

Component	**Objective of evaluation**
Scheduling	• Correctness of a feasible schedule in terms of completion time with respect to the desired time. Evaluation will follow the classical metrics for this class of scheduling heuristics; either in term of *makespan*, the execution completion time between the first and the last tasks or in term of throughput, tasks execution per time unit.

Qualitative and quantitative metrics

Component	**Qualitative and quantitative metrics of satisfaction**
Scheduling	We will test with the scheduling service against several parameters : – Scalability: by varying the number of working nodes, we will ensure that the heuristics used by scheduling service are not CPU intense and can still give a sufficient quality of service even when the number of nodes exceeds several hundreds (a excellent objective would be to reach 1000 nodes) – Workload: by varying the size of tasks, we will ensure that the scheduling service can cope with a wide range of parallelism grain. Our plan is to use a synthetic application where the following parameters can easily be set : number of tasks, task execution time, I/O operation, size of input and output parameters
Interface	– Satisfaction and correctness o Interface will be evaluated according to its integration within the grid4all framework. Through constant collaboration during the integration phase, we will ensure that the interface is coherent and well-defined.

Environment assumptions

Component	**Test bed**	**Test data**	**Environment conditions**
Scheduling	Grid5k and the gdx cluster	Test data will be based on the use scenarios derived from WP4;	We will use about a thousand machines on Grid5000 to test the scheduler with various workloads. To deploy the experiments, we will use OAR (the batch scheduler on Grid5000) to reserve resources and the software Tat-tuk to deploy our experiments. On Grid 5000, it is expected to use a thousand nodes distributed on more than 10 clusters, which will test the ability to cope with a wide deployment.

Expected functionality and test clients

Component	**Client (target user of functionality)**	**Test method**	**Expected functionality (from Grid4All)**
Scheduling	N/A	Unit tests using the JUnit framework	None
Interface	N/A	idem	None

5. Evaluation plan for WP3

5.1 Major functionality provided

WP3 software consists of three components.

Semantic Store (SS). The Semantic Store enables the development of collaborative applications, where users connected at different locations and/or different times create, view and update shared documents. The main implementation and validation of SS is the Telex middleware, which provides applications with replication and consistency services based on the Action Constraint Formalism (ACF). Telex interfaces with the underlying VOFS to store, replicate and retrieve application documents. There is also another implementation of SS based on the APPA P2P system which is useful to support a simpler, yet not as powerful, alternative to ACF. APPA uses a reconciliation algorithm based on operational transforms, called So6 based on the SOCT algorithm [2] and readily available as Open Source Software [1] and a Key-Based Timestamping Service (KTS) [5]. KTS interfaces with a Distributed Hash Tables (DHT) but could also use the VOFS in the future.

VO-aware File System (VOFS). The VO-aware file system provides Virtual Organisations (VOs) with a service of a distributed file repository formed by contributions of VO members. The file repository is created by aggregating different file systems, including those created by the DFS+VBS. Since VOFS files are VO resources, VOFS must interface with the VO management services to obtain and implement VO mechanisms and policies over the underlying file systems, especially for security (through the VO membership services).

Virtual Block Store (VBS). The Virtual Block Store layer aggregates storage from distributed contributors in a single virtual storage pool, in a peer-to-peer manner. Storage is considered an independent resource from files in the DFS architecture and VBS has the role to manage storage resources and provides them to the DFS layer.

The following table describes the packaging of each component. It also lists the external software used and the possible dependencies on plate-forms.

Component	**Interface technology**	**Software packaging**	**External software**
SS	Java API	.jar files	jgrapht.jar library (LGPL) for Telex, SO6 [1] for APPA
VOFS	POSIX File System user interface, Java API	Self-contained directory structure, .jar files	FUSE, Python v2.5, XACML library from Sun Microsystems, WebDAV from Apache
VBS	Python 2.5 module	Self-contained directory structure	Python 2.5

The following table describes when and how the components and their API will be made available to other WPs.

Component	API published	Software available	How/where
SS	2007/05	2007/10	http://gforge.inria.fr/projects/telex2/ http://gforge.inria.fr/projects/appa/
VOFS	2007/06	2007/12	http://gforge.inria.fr/projects/grid4all/
VBS	2007/04	2007/9	http://grid4all.cslab.ece.ntua.gr/

Note: SS software will be available to start developing applications in October 2007. A stabilised API and an initial prototype will be delivered in June 2008. VOFS and VBS software will be available for testing in November 2007. Their API will be stabilised and published at the 18th month consortium meeting in December 2007.

5.2 Semantic Store

5.2.1 Objective of evaluation

The SS middleware will be evaluated for:

- *Functional correctness:* A suite of functional tests will check that Telex conforms to the specification of the SS API. A particular attention will be paid to the correctness and quality of generated schedules, whether local or global (committed). To this end, a tool will automate the execution of a pre-defined set of update scenarios and check output schedules.
- *Adaptability and self*-behaviour*: Tests will be conducted to check that both Telex and APPA handle dynamic changes in the set of users editing a document without any functional or performance impact.
- *Performance:* Performance tests will measure the quantitative metrics defined below and evaluate the scalability of Telex and APPA with respect to the number of users that collaboratively edit a document.

Tests and evaluation of Telex will be conducted in three phases. The first phase will consist in testing the functionalities of the Telex middleware alone. During this phase, the VOFS services that Telex uses will be emulated through software developed ad hoc. The test of distributed functionalities, such as replica reconciliation, will be performed by emulating several virtual nodes on a single computer. The second phase will be dedicated to the integration of Telex with the VOFS. The tests will be performed on a small test bed involving a few computers on a local network. They will cover all aspects of the Telex – VOFS interface: file replication, update notification, message passing between Telex peer instances, membership based access control. The third phase will consist in performance tests of Telex over the VOFS.

The primary reason for having the APPA implementation of SS in addition to Telex, besides validating an alternative approach to ACF, is to study the feasibility of extending an existing collaborative application with SS functionality which is a hard problem. The existing application is Xwiki [7], a client-server wiki system which is used intensively for collaboration among small enterprises or within large organizations. Tests and evaluation of Xwiki/APPA will be conducted in three phases. The first phase will test Xwiki basic functionality over a small number of workstations connected by a local network, each workstation running KTS over the OpenChord DHT [6]. The second phase will test Xwiki complete functionality on the Grid5000. The third phase will include performance tests.

5.2.2 Qualitative and quantitative metrics

Telex will be evaluated according to two sets of quantitative metrics. The first set of metrics relates to local, low-level operations. They will be measured through micro-benchmarks run on a single computer. These metrics are:

- read and write fragment[1] *latency* and *throughput*
- fragment *size*, in memory and on disk
- schedule computing *CPU-time*

The second set of metrics relates to global, high-level operations. They will be measured by running a high-level benchmark on a large-scale test bed. Theses metrics are:

- *[first site] convergence latency*: the minimum time for [the first site] all sites to reach a committed state,
- *communication complexity*: the number and size of messages for all sites to reach a committed state.

5.2.3 Environment assumptions

Small-scale experiments will be conducted on clusters at INRIA. Large-scale experiments will use the Grid5000 test bed. Low-level functional tests will use scenario scripts developed manually as input. Performance tests will use one the OO7 [8], TPC-W [9] or Fages' benchmark [10]. Test data is part of the specification of these benchmarks.

5.2.4 Expected functionality and test clients

Telex and APPA do not need any service or functionality provided by other WPs.

Telex will be tested and evaluated through two client applications. The first application is a shell providing a command line interface to the Telex API, which will be used for functional tests. The second application is one of the benchmarks listed above.

APPA will be tested and evaluated using the Xwiki application. One major challenge is to make Xwiki P2P without any impact on Xwiki code base.

5.3 VO-aware File System

5.3.1 Objective of evaluation

The VOFS will be evaluated for:

- *Functional correctness:* The system must be tested for *conformity* to its own design and to the requirements of the systems and standards it implements, mainly the file access APIs. It is also necessary that the system is tested for the *safety* of user data. The tests will include common and extreme operational conditions, especially heavy load.
- *Behaviour when subject to load and failures:* We plan to simulate an environment which is very similar to reality (for example nodes can fail) and use VOFS component in this environment and test it.

1 a fragment is the update unit of Telex; it consists of set of related actions and constraints.

- *Performance:* Three aspects of performance will be tested. *Responsiveness* of the user interface, *throughput* of operations and *scalability*.
- *Adaptability:* In presence of VO evolution (growing of VO), nodes might join and leave VO very frequently. VOFS should tolerate churn and provides its functions with good performance. It also replicates services such as metadata management and mount server to achieve high availability and self management of replication. Replica management is transparent to the clients.
- *Self-* behaviour:* We will test different self-* properties of VOFS like self-management, self-tuning and self-healing in case of the failure of nodes.
- *Ease of use from point of view of targeted user:* VOFS provides standard POSIX file API that allows it to be used by many applications. VOFS also provides client program with GUI to make it easy to perform operations like exposing and mounting for the user.

5.3.2 Qualitative and quantitative metrics

A suite of micro-benchmarks that target each basic operation will give a metric of the raw performance capabilities of the system and offer an upper bound for the performance evaluation of more complex, higher level procedures. Apart from micro-benchmarks, standard file system testing tools and applications will be used to test the performance of traditional operations. For the unique operations, benchmarks will be created according to the system's usage by the applications. The basic metric in all these tests is the throughput and latency of operations.

A basic list of basic operations to be tested is:

- Rate and latency of opening / closing files
- Rate and latency of read / write operations on cached files
- Rate and latency of read / write operations on remote files

The *responsiveness* and *ease of use* qualities will be evaluated empirically since they need subjective and complex judgements. Since the casual operation of the system involves user activity the best way to make the evaluation is through the use of an actual deployment of the system in a real environment, supplemented with synthetic tests for specific cases.

5.3.3 Environment assumptions

VOFS will be tested on the Grid4All test bed. This includes the Grid5000 project, PlanetLab, clusters at SICS, or any other Grid system that will be used in Grid4All. The basic data in our test bed will be the information about VO, information about users of VO and information about their resources (files) that are part of VOFS. The main information in VOFS will be data in form of different types of files with different sizes. During the test of VOFS, users should form a VO, create a VOFS and use its functions (for example exposing files, mounting, caching, replication etc.). Nodes might join, leave or fail during test period. The test should perform different checks to be sure of correctness of the operations and to check the performance of the system. Another functionality of VOFS that should be checked is availability of data with respect to churn. Also we will test the VOFS component by creating large VOs with large number of users to test scalability.

5.3.4 Expected functionality and test clients

VOFS is a service for a VO. Therefore to test it we need to be able to create a VO and manage it. This creates a dependency on VO management services from WP2. An opportunity for collaboration exists with WP1, if the DFS substrate of VOFS uses their overlay services.

VOFS is directly usable as a mountable file system, therefore a special test client is not needed. The system can be directly tested by applications, such as text editors and document viewers. For the purpose of testing and demonstrating the features of the system, an application demo will be developed and included in the distribution.

5.4 Virtual Block Store

5.4.1 Objective of evaluation

The Virtual Block Store (VBS) components will be evaluated for:

- ***Correctness of specified functionality***.

  Correctness will be tested in the form of *conformity* with the specifications and in the form of *safety* of user data.

  The system must conform to its own design and must also have consistent behaviour across all platforms it supports.

  The safety aspect of correctness is particularly important because loss of user data is much more serious than the inconvenience that is caused by other errors.

  Correctness will be tested both under normal and extreme operational conditions, especially under heavy load.

- ***Performance.***

  The following aspects of performance will be evaluated:

  1. *Throughput and latency*

  The higher rate of operations executed by the system and the faster their response, the more processing power the user commands. Therefore, the throughput and latency of the various operations will be evaluated as a basic performance test.

  2. *Scalability*

  The distributed nature of VBS requires that the system must be able to operate under a distributed load. The limits of the system scalability will be evaluated and compared to the requirements of the applications.

5.4.2 Qualitative and quantitative metrics

The VBS will be deeply integrated into the DFS substrate of VOFS, therefore its evaluation will be done through the evaluation of the VOFS. However, for the purpose of *performance* and *safety* evaluation, micro-benchmarks will be run that test for basic functions. A basic list is:

- Rate and latency of allocations / deallocations of storage
- Rate and latency of store / retrieve operations
- Rate and latency of transfer of big data chunks

5.4.3 Environment assumptions

The system will be tested in a real-world deployment, as part of VOFS and DFS installations in the Grid4All test beds. Test data will not be directly used for the evaluation. Running application-level tests indirectly subjects the system into rich testing scenarios. If needed, new application-level tests may be contributed so that all features may be tested.

5.4.4 Expected functionality and test clients

The Virtual Block Store (VBS) does not have any dependencies. However, integration with the market-based resource management infrastructure in WP2 is desirable. VBS was designed and developed as an integrated component of the DFS (and therefore VOFS) architecture. Therefore, general testing and demonstration of VBS can be made through VOFS.

6. Evaluation plan for WP4

6.1 Major functionality provided

The main objective of this work package is to demonstrate how the remaining subsystems developed in the project can work together to support a number of applications that are representative for a list of relevant usage scenarios. These applications will be used to understand the complexity imposed by this environment on applications, propose and negotiate API features between applications and underlying services to facilitate the development of applications, and produce application level traces and benchmarks to evaluate and validate the Grid4All environment.

Deliverable D4.1 defines realistic scenarios where Grid4All is applicable describing user requirements, operational and functional requirements for the infrastructure (infrastructure requirements). Several cases are evaluated: collaborative learning, domestic users, small enterprises, communities with a total of 12 different scenarios presented.

Deliverable D4.2 describes a programming model suitable for the development of Grid4All applications and describes four applications representative of several of the scenarios described in D4.1, and their infrastructure requirements. Therefore we refer to D4.2 for details on the applications and their relation to infrastructure functions and API.

The main four applications for Grid4All validation are described in the following table:

Component	**Description**
eMeeting (EM)	This application is intended to support users interacting (at the same time, in the same or different places) in separate virtual rooms, using text, voice and video. It needs the following main features from the infrastructure: (a) distributed storage, for the in and out operations regarding the storage of the polling, slides, and log files; (b) Virtual Organization management, to retrieve information about rooms and participants, and grant or deny access to each user; (c) Service discovery, to allow the application to discover, select and use different services or tools as part of the eMeeting environment; (d) Resolution of potentially conflicting dating and/or calling for a meeting. The application is formed by three components: two end user oriented components: the administration component, the front-end component, and the communications service.
Collaborative Network Simulator Environment (CNSE)	This application is intended to allow educators in the Computer Networks domain to design activities for their students that are intensive in network simulation, profiting from the computational and storage resources available in a grid. It needs the following main features from the infrastructure: (a) storage, retrieval and removal of files (WP3, task 3.1); (b) service discovery infrastructure to locate application level services. (semantic information system proposed in WP2, task 2.3); (c) user credentials from the membership service (Virtual Organizations, WP2); (d) deployment and monitoring services to achieve self-* properties (WP1). The application is formed by three components: two end user oriented components: the management component, the participation component, and a number of supporting application-level services.

Component	Description
Collaborative File Sharing (CFS)	This application is a 'shared workspace' system which supports document sharing, asynchronous discussions, task list management and a private address book. It is related to all scenarios with needs in sharing information among users. It needs the following main features from the infrastructure: (a) mechanisms for supplying and storing data in a reliable way (DFS, WP3); (b) mechanisms for detecting and solving conflicts (Telex, WP3); (c) user credentials from the membership service (Virtual Organizations, WP2). The application is based on a single application related component, with as many active instances as active participants.
Shared Calendar (SC)	This application allows participants to share their own agenda, invite others to meetings, or alter other people's agendas, supporting disconnected operation. It needs the following main features from the infrastructure: (a) access control and membership identification (WP2, WP3); (b) most functions provided by Telex (WP3); (c) Replication and communication support (WP3, WP1); (d) document information and group awareness (WP2); (e) resource management services (WP2). The application is based on a single application related component, with as many active instances as active participants.

The following table describes the packaging and dependencies of each component. It also lists the external software used and the possible dependencies on platforms.

Component	Interface technology	Software packaging	External software
EM	Java API Web Services WSDL	.jar files	Flash Media Server, Flash runtime (embedded on web browser)
CNSE	Java API Web Services WSDL	.jar, .gar files	NS Network Simulator NAM Network Animator X virtual frame buffer (Xvfb) x11vnc
CFS	Java API XUL XPCom	.xpi file (Mozilla Cross-Platform Install)	Mozilla Firefox
SC	Java API	.jar files	

The following table describes when and how the application components and their API will be made available to other work packages.

Component	API published	Software available	How/where
EM	2007/12 (M18) [D4.3 in M24]	2008/1 (internal) 2008/5 (M24) [D4.4 in M24] 2008/5 (M24) (final) [D4.5 in M30]	http://gforge.inria.fr/projects/grid4all/
CNSE	2007/12 (M18) [D4.3 in M24]	2008/1 (internal) 2008/5 (M24) [D4.4 in M24] 2008/5 (M24) (final) [D4.5 in M30]	http://gforge.inria.fr/projects/grid4all/
CFS	2007/12 (M18) [D4.3 in M24]	2008/1 (internal) 2008/5 (M24) [D4.4 in M24] 2008/5 (M24) (final) [D4.5 in M30]	http://gforge.inria.fr/projects/grid4all/
SC	2007/12 (M18) [D4.3 in M24]	2008/1 (internal) 2008/5 (M24) [D4.4 in M24] 2008/5 (M24) (final) [D4.5 in M30]	http://gforge.inria.fr/projects/grid4all/

6.2 Evaluation

This section presents the validation and evaluation objectives at the level of each functional component. The objectives of the functional component level evaluation are:

- ✓ To verify the *functional correctness* of the implemented code before integration within a more complex use case. This can be done with a suite of tests data to check conformance to the offered API.
- ✓ To understand the *functional behaviour* of each isolated component, and the effect of limited interaction with the environment: end users (real or simulated) and system environment (with dummy external components).
- ✓ To evaluate the *adaptability and self-* behaviour* in presence of changes on the characteristics of resources, on demand and events such as node or network failures.
- ✓ To derive concrete indicators that may be used to suggest correct configurations of systems within a demonstrator that incorporates real-world assumptions on scale

The objective of evaluation for each application is described in the following table:

Component	Objective of evaluation
EM	- Adaptability and self-* behaviour - Functional correctness - Scalability in terms of Rooms and automatic deploy and configuration.

Component	Objective of evaluation
CNSE	- Scalability of in terms of participants - Performance in terms of time required to complete simulations
CFS	- Functional correctness - Application behaviour on a dynamic environment - Application behaviour in presence of conflicts - Ease of use of the user interface
SC	- Functional correctness - Telex validation and evaluation

6.2.1 Qualitative and quantitative metrics

The application's components will be evaluated one by one according to mostly quantitative metrics in respect to scale where the units or metrics will be defined for each application or subsystem. In general terms, number of actions performed respect to (a) changing demand, (b) changing amount of resources available, (c) changing failure rates (churn). In addition, each application component can be tested in respect to critical situations specific for each component: effect of certain sets of operations that produce a particularly critical behaviour of the component (e.g. sets of operations that produce conflicts to measure the performance in that situation).

The qualitative metrics related to the cost of the system (as a service composed of multiple coordinated components): number and volume of messages, effect of internal synchronization among components in response time, time and messages to reach a stable state.

The following table describes the metrics for each application:

Component	Qualitative and quantitative metrics
EM	- Measure of time to adapt to changing load (time for allocation of resources, deployment of new instances and starting of these new instances).
CNSE	- Measure of notification time (time elapsed since a user generates an event until it is delivered to his collaborative partners) to evaluate scalability in terms of participants - Measure of time required to complete simulations
CFS	- Measure of scalability in operations performed concurrently by many users: generating a new file version and solving a update conflict - Ease of use of the application
SC	- Code complexity - Ease to use from the targeted user point of view - Scalability in term of participants and events Convergence latency: the minimum time for [the first site] all sites to reach a committed state.

6.2.2 Environment assumptions

Small-scale experiments will be conducted on clusters at UPC. Large-scale experiments will use the PlanetLab or the Grid5000 test bed. Low-level functional tests will use scenario scripts developed manually as input. The data for testing will be a set of operations that simulate the interaction between the user and the application.

The following table shows environmental assumptions for each application:

Component	**Test bed**	**Test data**	**Environment conditions**
EM	Network running Grid4All middleware	Different VO configurations and user sets	Variable number of nodes for dynamic deployment
CNSE	Network of Pentium IV class machines with up to 25 nodes	Appropriate simulation scripts describing network scenarios with interest for Computer Networks education	Variable number of nodes, variable load of nodes.
CFS	Network running Grid4All middleware with a bounded dynamism.	Set of operations related to the same and to different items to check behaviour under conflicts	Fixed or variable load on nodes, variable node failures and network split (Testing off-line operation and join)
SC	- Low level: emulating several virtual nodes on a single computer. - High level: Grid4All test bed (Grid5000, PlanetLab , ….)	- Use case scenario developed manually: - Generated operation scenarios - Network simulation scenarios	- Variable number of operations/nodes - Volatility of nodes: dynamic leave and join. - Injection of failures.

6.2.3 Expected functionality and test clients

The applications will be tested, firstly, in a stand-alone way and then, integrated with other Grid4All services. The first set of tests check correctness of the interface and the local software. The second set of tests check integration with services part of other work packages.

The following table describes the test methods that will be used, and the targeted users (clients) of each of the components. Individual test of component will develop test clients which are representative of the real clients. The expected functionality represents the other Grid4All software on which each component is dependent for a complete integrated evaluation (not needed in the scope of the local evaluation).

Component	**Client (target user of functionality)**	**Test method**	**Expected functionality (from Grid4All)**
EM	– Domestic user, Students – Administrator for deployment and configuration	– Unitary tests for correctness (Java clients) – Unitary tests (FMS)	Overlay services (WP1) and VO management framework (T2.1) for the deployment. Distributed File System (WP3)

Component	Client (target user of functionality)	Test method	Expected functionality (from Grid4All)
CNSE	– Educator: create groups of students (group context service) – Student: Launch simulations and visualizations, and collaborate (simulation service, shared repository service, awareness service, visualization service, group context service)	– Unitary tests for correctness (Java clients)	Distributed file system proposed in WP3, task 3.1; and semantic information system proposed in WP2, task 2.3
CFS	– Writer: New file version uploaded, new message post, etc. – Reader. – Administrator: Data management	– Unitary tests for correctness (Java Clients)	Semantic Store (WP3) and Distributed File System (WP3)
SC	– Domestic user – Enterprises	– Java clients	- VO Membership service provided by WP2

7. Use scenarios and applications

This section presents a set of use scenarios and applications that will be used as yardsticks for the evaluation of the middleware and services developed within Grid4All. The generic use cases proposed in the description of work for Grid4All have been explored in detail for some selected scenarios as described in D4.1:

- Collaborative learning: “Using Asynchronous Multimedia Collaborative Tools and Grid resources to support Collaborative Learning”, “Collaborative simulation of computer networks”, “Distributed Collaborative Software Project Development” and “Live tutorized online sessions”.
- Domestic users: “Multimedia and entertainment spaces”, “Using grid resources and digital home health services to fight various diseases”.
- Small and medium enterprises: “Live sessions using collaborative tools”, “Collaborative software development”, “Earthquake prediction”.
- For-profit and non-profit communities: “Communities”, “Emergency handling”, “Shared virtual libraries”.

From each of these scenarios, user requirements and infrastructure requirements have been derived. In generic terms, there are three broad scenarios: learning (an arrangement with the purpose of learning), domestic users (a collection of individuals with diverse and personal purposes), and organizations (an arrangement with a specific purpose) including SME and for-profit and non-profit communities.

The choice of applications has been influenced by the experience and expertise and background contributions of the partners, the relation with the scenarios in D4.1, the interest of the application domain and the potential to exercise and expose the features of the work in other work packages. Four applications have been selected where each is described in detail in the deliverable D4.2:

- The eMeeting (EM) application will support users to interact in separate virtual room, using text, voice and video.
- The Collaborative Network Simulator Environment (CNSE) will allow educators in the Computer Networks domain to design activities for their students that are intensive in network simulation, profiting from the computational and storage resources available in a grid.
- The Collaborative File Sharing (CFS) application will allow users to have a unified view of a distributed file system, and to share and version files on it.
- Finally, the Shared Calendar (SC) application will allow participants to share their own agenda, invite others to meetings, or alter other people’s agendas, supporting disconnected operation.

These applications depend on the infrastructure in different degrees. All rely on distributed storage and virtual organization support provided by the Grid4All services. Besides, some make use of communication, resource management and allocation and conflict/resolution mechanisms. Some applications, as described in the previous section (CNSE and EM), include application-specific services that have to be deployed and managed, whereas SC and CFS are peer-to-peer applications instantiated for each participant making use of the remaining Grid4All services (Telex, DFS, etc.)

In respect to the applicability to real settings, according to the Annex 1 of the contract:

> “Even though within the current proposal, testing and demonstration is planned on simulated environments on test Grids, it is our intention to plan pilot studies in a real setting involving schools, local community centres and parents of school children.”

Therefore as part of the final integration (D4.5: *Integrated prototype of Grid4All applications*, at M30), and Evaluation (D5.3: *Final integrated proof-of-concept implementation and evaluation report*, at M30) all the applications will be described in the context of potential pilot studies in real settings particularly in the learning context, but also in the generic context of collaborative work where EM, CFS and SC could be incorporated as well as parts of the CNSE.

Thus, also according to the Annex 1 of the contract:

> “These applications will be used to (1) show examples of relevant uses for small communities such as families, schools and SMEs of the kind of Grids that Grid4All intends to enable; (2) find ways to design or redesign applications based on the infrastructure proposed by Grid4All; and (3) provide an environment for the validation of the Grid4All model and implementation”.

The following table has chosen three broad scenarios and presents the specific environmental conditions, functional and non-functional expectations and the relevant metrics of satisfaction for each of the scenarios.

Scenario	Learning	Domestic	Organizations
Objective	Perform learning activities that require collaboration among students and grid computing for computing intensive tasks (e.g. schools and learning oriented communities)	Perform tasks that require on-demand computing for media management and interaction with other people (e.g. families, communities)	Arrangement of people and computational resources with a shared goal (e.g. SME, for-profit and non-profit organizations)
Scale - typical # active participants - number of nodes	 15-100 10-1000	 2-50 2-100	 10-1000 10-1000+
Test bed	School PC + remote web clients + additional servers	User's PC [+ additional servers]	User's PC [+ internal and external servers]
Test situations	One group of students (2-6 students) One course (3-20 groups) One school (5-20 courses) A scholar community (10-1000 schools)	A group of friends (2-20) A familiar group (5-50)	A small SME (10-20 persons) A non-profit association (20-1000) A neighbourhood (100-1000 persons)
Environmental conditions	Variable # of nodes Variable load of nodes Nodes can fail or be down on a daily basis Changing network conditions but minimum quality can be assumed (School/Educational network)	Variable # of nodes Variable load of nodes Nodes can fail or be down frequently (hours, minutes) Diverse and changing network conditions Potential off-line work	Variable # of nodes Variable load of nodes Nodes fail or be down frequently (hours, minutes) Diverse and changing network conditions Potential off-line work
Contention	Effect of ordered tasks (waiting time due to coordination) Lack of sufficient resources (simultaneous demand from students) Due to competition among separate groups, classes, schools	Effect of ordered tasks Lack of sufficient resources Conflicting actions Due to competition among overlapping or separate tasks, communities	Effect of ordered tasks Lack of sufficient resources Conflicting actions Due to competition among overlapping or separate tasks, organizations
Self-adjusting behaviour	Adaptation to varying (peaks of) demand Adaptation to varying resource availability	Adaptation to varying (peaks of) demand Adaptation to varying resource availability	Adaptation to varying (peaks of) demand Adaptation to varying resource availability

	Adaptation to failures Adaptation to configuration	Adaptation to failures Adaptation to configuration	Adaptation to failures Adaptation to configuration Self-protection
Applications	Mainly CNSE, EM But also CFS, SC	Mainly CFS But also EM, SC	Mainly CFS, EM, SC
Quantitative metrics	Time required to adapt (port) applications to the Grid4All environment Efficiency (overhead in using framework) Number of successful tasks performed on time (compared to baseline environment) Ratio completed/initiated tasks Statistics of response time of tasks Adaptation to environmental changes: ratio of events handled by (passed) application / total events with varying rate of changes Adaptation to scale: application level performance with varying scale	Idem	Idem
Qualitative metrics	Designer/developer/user satisfaction Generality of framework (effort of adapting diverse applications to scenario) Effect on usability (based on survey)	Idem	Idem

8. Conclusions

Evaluation is aimed at assessing both the performance and usability of Grid4All software and is carried out during different phases of the software releases. Evaluation of solutions to given problems is an important question in conducting research aiming technological applications. We need to choose not only a set of measures for any studied solutions, but also choose a context and its representative environmental conditions which prescribes how the measures are to be evaluated for a given set of solutions. The environment needs to be abstracted since it is impossible to consider all possible details that are present in real-world applications. Hence local evaluation of released software through empirical approaches based on simulations and generated data sets is an important step.

Different aspects of evaluation criteria are relevant within Grid4All:

- Quantitative measures of satisfaction such as performance criteria, computational complexity, and scalability apply to well-defined measurable situations,
- User satisfaction criteria applies to situations where the usability and pertinence of the provided solutions are being evaluated,
- Developer and designer satisfaction criteria applies since a large part of the middleware and developed services have taken a framework based approach that allows specific software components to be specialized or adapted to new situations.
- Non-functional aspects such as ability to reconfigure and adapt to changes in environment

The present document identifies the first plan for evaluation. Software modules will be individually evaluated as and when possible through usage of generators to model data environment, traces representative of the real environment and when the case may be provide benchmarks of comparisons between proposed solutions and compare to competing solutions. The use scenarios and applications that will be used for the common assessment objectives for integrated middleware and service stacks.

The aim of the evaluation is to derive interesting properties of the system and where possible derive quantitative measures of satisfaction for these properties:

- Adaptation refers to how the system reacts to changes in the environment and how rapidly it reacts so as to restore the system to expected measures of behaviour.
- Self-management refers to the capability of the system to autonomously react to changes without requiring external intervention.
- Scalability refers to the ability of a system to function within expected levels, even when its size, measured by system-specific parameter, increases.
- Pertinence, usability and user satisfaction which are difficult to characterize, but are important criteria for the assessment.

A real environment is the actual deployment of the integrated solution on the real Internet. This level may not be reachable within the scope of the project. Hence the next step will focus on the satisfaction of the defined use scenarios within a test environment by deploying the middleware and services on a test-bed such as Grid5000.

9. References

[1: SO6] http://dev.libresource.org.

[2: SOCT] M.Suleiman, M. Cart, J. Ferrié: Serialization of concurrent operations in a distributed collaborative environment. ACM SIGGROUP Conference on Supporting Group Work (GROUP), 1997, 435-445.

[3: CVS] Per Cederqvist. Version Management with CVS. Network Theory Ltd. 2002.

[4: SVN] CollabNet. Subversion, (Septembre 2005). http://subversion.tigris.org/.

[5: KTS] Reza Akbarinia, Esther Pacitti, Patrick Valduriez. Data Currency in Replicated DHTs. SIGMOD Conf. 2007. 211-222.

[6: Open Chord] http://open-chord.sourceforge.net.

[7: Xwiki] http://www.xwiki.com.

[8: OO7] Michael J. Carey, David J. DeWitt, Jeffrey F. Naughton. The OO7 benchmark. SIGMOD Record (ACM Special Interest Group on Management of Data), 1993.

[9: TPC-W] www.tpc.org/tpcw.

[10: Fages'benchmark] F. Fages. A constraint programming approach to log-based reconciliation problems for nomadic applications. In 6th Annual W. of the ERCIM Working Group on Constraints, Prague (Czech Republic), June 2001.