Project no. 034567

Grid4All

Specific Targeted Research Project (STREP)
Thematic Priority 2: Information Society Technologies

Deliverable D 4.6 (revised):

Grid4All: Democratic Grids — Scenario and Architecture

Due date of deliverable: June 2008.

Actual submission date: 20th June 2008.

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

Organisation name of lead contractor for this deliverable: All partners

Revision: Deliverable

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level

PU	Public	√
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)	

Contents

List of Figures

Abbreviations and acronyms used in this document

Abbrev./acronym	Def.	Description
ACF	p. 26	Action-Constraint Framework
ACL		Access Control List
ADL	p. 14	Architecture Description Language
API		Application Programmer Interface
BPEL		Business Process Execution Language
CFS	p. 28	Grid4All Collaborative Filing System
DCMS	p. 11	Distributed Component Management Service
DFS	p. 23	Filing layer of VOFS
DHT	p. 2	Distributed Hash Table
Fractal	p. 6	Component model used in Grid4All
GRIMP	p. 18	Grid4All market place
iEARN	p. 2	Educational and Research Network
MIS	p. 21	Market Information Service
NGO		Non-Governmental Organisation
OGSA		Open Grid Services Architecture
OSGI		Open Grid Services Infrastructure
OWL-S		Semantic Markup for Web Services
PAP	p. 16	Policy Administration Point
PDP	p. 16	Policy Decision Point
PEP	p. 16	Policy Enforcement Point
PR	p. 16	Policy Repository
SA	p. 10	Sensor-actuator loop
SC	p. 28	Grid4All Shared Calendar
SIS	p. 18	Grid4All Semantic Information System
SOAP		Simple Object Access Protocol
SPARQL		Query Language for RDF
Telex	p. 25	Grid4All Semantic Store
URI		Uniform Resource Identifier
VBS	p. 24	Storage layer of VIFS
VO	p. 8	Virtual Organisation
VOFS	p. 22	Grid4All VO-oriented distributed file system
WS-Choreography		Web Services Choreography
WSDL		Web Service Definition Language
XACML	p. 16	eXtensible Access Control Markup Language

Grid4All list of participants

Role	Part. #	Participant name	Part. 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

Abstract

A grid virtualises resources in order to share them within virtual organisations (VOs). Grid4All promotes the concept of a *democratic* grid, enabling grid access to VOs such as schools, families, NGOs or small businesses. In this paper, we examine a typical scenario for educational VOs, and derive the Grid4All architecture from this scenario. The Grid4All infrastructure, based on a component model, supports a number of properties required in the democratic grid: usability, self-administration, dynamicity, and security. Grid4All supports novel and collaborative applications. Grid4All VOs support collaboration thanks to network-wide shared storage, federated file workspaces, a semantic approach to consistency, and collaborative task execution.

1 Introduction

Grid4All promotes the concept of a *democratic grid*, accessible to modest groups of end-users such as schools, families, NGOs or small businesses. The Grid4All design principles follow from our usage scenario and use cases, and include accessibility, usability, and minimal administration.

A grid is a distributed system that virtualises and aggregates resources exposed by separate institutions. A key concept is the Virtual Organisations (VO), a virtualised collection of users or institutions that pools their resources into a single virtual administrative domain, for some common purpose. Grid4All shares the requirements of the Open Grid Services Architecture (OGSA) [7, Section 2]. However, previous grid systems, such as OGSA, are designed for "big-iron" use cases, for instance commercial data centres, large-scale scientfic computing, etc. They remain complex and heavy-weight.

Our use cases, involving schools, learning institutions, families, and small businesses are very different, as documented herein. For instance, our scenarios are mostly collaborative, whereas remote execution of large computations is less central than in existing grids. In the area of data services, compared to previous grids, the Grid4All architecture provides for flexibility, usability, and support for collaboration within VOs.

In summary, our design is oriented towards usability, content sharing, collaboration, and flexibility. Accordingly, Grid4All extends on the current state of the art of grid systems, focusing on a small number of challenges:

- **Usability for lightweight, flexible and dynamic VOs.** Many VOs will be short-lived (minutes, hours or days), but some will last very long. Most will be small (a few tens of users), but some may be very large. Users will join and leave, and resources will be added or become unavailable, in an unpredictable fashion. The manual effort to set up or terminate a VO should be extremely low. The system should respond appropriately, minimizing manual intervention, to unplanned events.
- **Novel and collaborative applications.** Existing grid systems, designed for large-scale scientific computation, focus on massive remote execution services, and data management is relatively primitive. Grid4All provides novel support for remote execution, targeting home and school users. However its greater focus is to support novel collaborative applications, such as distributed authoring tools, e-learning environments, or decision-making.
- **Application management in dynamic VOs** Not only must the VO be kept in a robust and coherent state in the face of the greater dynamism in democratic Grids but also the service/applications running within the VO. We need to provide the basic tools and monitoring/control infrastructure to enable application programmers to develop self-managing applications for Grid4All.
- **Availability, scalability and decentralisation.** The size of a VO in an educational institution might range from ten users to a few thousand. However, a popular VO may grow unpredictably. Even if individual VOs are small, there may be many, and they may have a large geographical distribution. Therefore, the Grid4All system should be able to scale seamlessly over a wide range of sizes and of geographical and administrative extents. The volatility of resources makes achieving availability challenging. Accordingly, the architecture avoids any single point of failure, or centralised performance bottleneck.

To meet these challenges, the Grid4All architecture is based on a peer-to-peer overlay infrastructure, enables autonomous management, and provides federative and collaborative data services:

- **Overlay infrastructure** We use peer-to-peer (P2P) networking techniques, e.g., self-organizing overlays and Distributed Hash Tables (DHT), which tolerate high rates of unplanned resource arrival and departure (churn). They scale well thus supporting VO evolution. The overlay infrastructure holds the VO together in dynamic environments assuring that individual resources, members, and components can be discovered, monitored and controlled.

- **Autonomous management.** To ensure usability in a dynamic environment, Grid4All aggressively promotes autonomous management control loops at all levels. Self-management is considered along four axes: self configuration, self tuning (or self optimisation), self healing and self protection. This approach is often called *Self-** (pronounced "self-star"). Within a single component-based application or service this is enabled by a distributed component management service built on top of the overlay infrastructure.

- **Federative and collaborative data services.** To support flexible and dynamic VOs requires a novel approach to information sharing. The Grid4All data storage architecture provides three different levels of flexibility. (i) VO users can pool disk space over the network: either offered to each other from their own disks, or acquired on an open market. (ii) A VO has a associated virtual file system, a flexible federation of files names and file contents, exposed by VO users to one another. (iii) A semantic middleware layer allows users to update shared data in a flexible manner. Users may share content in either on-line or off-line mode; the semantic store resolves conflicting updates according to data semantics.

This paper proceeds as follows. Section 2 documents the overall scenario and example use cases that dictate the Grid4All design. Section 3 sets forth the principles of our architectural design. Section 4 describes the core VO support architecture. In Section 5, we describe the Grid4All discovery and market services. Section 6 focuses on VO mechanisms facilitating data-centric collaboration within a VO. Finally, Section 7 concludes.

2 Educational Usage Scenario

The Grid4All concept could be applied to a number of different use cases. In order to focus our design, we consider a high-level scenario in the educational area.

According to Jarke et al. [10], a scenario is a concrete usage example, within a a description of a context and for a purpose, and focusing on task interaction. Scenarios comprise the following elements [4, 5]:

- A setting: describing the starting scene for the described episode.
- A set of actors: human activities scenarios include many actors.
- Goals and objectives: actors typically pursue different goals or objectives.
- Actions and events: a scenario has a plot, i.e., a sequence of actions and events.

2.1 The setting

Our scenario takes its inspiration from iEARN, the International Educational and Research Network [9]. iEARN is a world-wide educational network of inter-university courses supporting the collaborative educational projects, within regional or world-wide university collaboration networks.

Our scenario involves a network of schools engaged in collaborative activities. A collaboration consists typically of a course or a course project. It may involve students and instructors from multiple sites, spanning different departments, schools, countries, time zones or cultures. The member institutions may be schools at any educational level (primary, secondary or university). Participants work either from home or at their school.

A collaborative network pools resources from different providers for its activities. The providers include the schools, the students, the parents, and outside partners. The resources may be physical or virtual, and include computational, storage and other resources.

Educational institutions already rely on specific digital tools, for instance to consult external experts, to perform digital experiments and simulations, to communicate with other students, and to work remotely. These tools should be available in the collaboration, and should be made aware of the collaboration context.

Collaborative work tools should allow a group of users (e.g., a class, or a set of instructors from different schools) to share computational or storage resources, to share and modify content together (e.g., to author a document collaboratively), and to make common decisions. However, the system must ensure that users access resources or information only if authorised, and only in an authorised way.

Collaborative groups are fluid; some are short-lived, some long-lived; over time, the membership of a group changes. Groups are likely to overlap: some users will belong to several groups; the same content can be viewed in different groups. Content belonging to a group should be isolated from other groups. Content must be persistent and modifiable (subject to authorisation).

The schools' existing infrastructure may vary significantly. For instance, the OLPC project [13] relies on an ad-hoc network of personal computers with no dedicated servers, whereas PlanetLab [14] collects consists of shared servers in a P2P manner but provides only best-effort quality service; Grid5000 [8] is similar but is a more stable environment with resource reservation. Despite the differences, networks remain unreliable. Churn (computers connect and disconnect without notice), failures, and service degradation are arbitrarily frequent.

To conclude, we summarise some of the challenges inherent in the Grid4All setting:

- The work environment is dynamic and unpredictable. Participants come and go. Load varies: just before an exam deadline, many students run resource-intensive activities simultaneously. Computing resources fail.
- Nonetheless, the system should remain manageable, in order to ensure some level of dependability. For instance a class project should not miss its deadline because of a glitch in the system. Otherwise, schools will be reluctant to use it.[1]
- The system must be simple. Manual administration cost must be very low. The system should adapt automatically to problems. Collaboration should not tax the schools' (usually very limited) support staff.
- A collaboration incorporates rules and policies. For instance the members may need to abide to pre-existing rules of the participants, or to make contributions, either monetary or in kind (e.g., computational resources).
- Participants may be in multiple locations: in the school, at home, in a library, on the move, or work in other cities or countries elsewhere in the world.
- Participants must be able to work remotely in disconnected mode. It is not reasonable to expect that all participants are connected all the time.
- The responsiveness of the system should not degrade despite partial failures, network problems, and disconnected-mode work.

2.2 Actors

End-users in this scenario include learners and educators. A learner participates in the collaborative learning situation with the aim of obtaining new knowledge and skills in relation with the learning activity (computer networks in our scenario). Learners may be enrolled in different classes or schools.

An educator (or a group of them proposing a certain project) is responsible for providing the learning context used by learners in the situation (e.g., questionnaires, simulation files, shared folders, etc.) as well as for answering the questions posed by learners during the realization of the situation.

The scenario involves actors other than end-users, e.g., providers of services or resources or application developers. A individual interested in some activity such as pupils or students, instructors, parents,

[1] A typical comment from an instructor: "What happens if your system fails when I am alone in my class with 24 teenage students? I prefer proven technology, even if it means reducing the scope of my activities. After all, I can rely on pen and paper, or on a simple slide projector."

families or friends may be willing to contribute resources. The local educational institution in which the participants in the scenario are enrolled is an important actor. Other educational institutions may be willing to provide their resources in return for in-kind payment (e.g., later access the resources owned by the local institution). Companies or individuals may also offer resources used in the scenario, in exchange for economic compensation.

2.3 Goals and objectives

2.3.1 Collaborative learning

Cooperative or collaborative learning is an instruction method based on students working together in small groups to accomplish shared learning goals [11]. Success depends on five essential "glue" components: positive interdependence, face-to-face promotive interactions, individual accountability and personal responsibility, teamwork and social skills, group processing. Group members should feel that they need one another, should be willing to help each other learn, and should have a personal stake in the success of the group. They should also have group skills and be able to make regular adjustments as needed by the group's current strengths and weaknesses.

Successful collaborative learning requires effective and appropriate implementation of student groups. Formal groups require planning the size and composition of the group with greater structure and a specific purpose (e.g., a particular task to accomplish); group membership should remain stable, ideally over a full academic term. Informal groups may be created and dismantled for a short term specific task, from minutes to a few days. Semi-formal groups may be as structured as formal groups, but have a shorter duration (weeks at most).

In general, groups should be heterogeneous. Students can be chosen randomly from an attendance roster or they can count off. Examples of this are:

- Groups of 3 students doing some joint activity during several minutes to one hour as part of a single course.
- Groups of 4-5 students doing a laboratory activity (e.g., computer networks lab) using their own resources or resources provided by the institution, lasting weeks or months.
- A project with N student's groups performing a collection of activities over a medium-long term.

2.3.2 Collaborative learning of networking by simulation

Simulators are nowadays usually employed with educational purposes in many Computer Network courses. Nevertheless, the actual use of simulators within the context of educational institutions is sometimes hindered by the following two problems. First, the computer resources available are sometimes not enough to carry out in a reasonable period of time the many simulations that can be launched at the same time by the students attending a laboratory session. This fact limits the complexity of the simulations that can be carried out and, as a consequence, the possibility of illustrating many network scenarios that could be of interest for students. Besides, the storage resources available are sometimes scarce. However, the amount of data generated by some simulations may be very large.

Again, this implies that this type of simulations cannot be carried in order to foster the reflections of students on many network scenarios. Within this context, we introduce a collaborative learning scenario in which students can launch simulations not limited in terms of the computer power required to carry them out or the storage resources needed to store the data generated thanks to the support provided by the computational grid.

These activities will benefit from an infrastructure such as Grid4All's provides, and particularly, group applications to negotiate, coordinate and create shared knowledge or shared information and group awareness to coordinate groups (group repository with documents, annotations and events denoting changes or awareness information and a shared calendar to articulate work and meetings), and support for building applications to support task-specific activities.

2.4 Actions and events

In this section, we describe the typical actions and events for our educational environment based on our experience with iEARN and university level collaborative learning projects.

A group of schools decide to set up a joint project on some topic (e.g., writing a multimedia presentation of some topic, such as “Performance evaluation of Internet protocols on ADSL links”). For that, several instructors have to specify all the details required to describe the activity in sufficient detail and determine the rules of participation. There will be a draft document describing the proposal that will be published in a web portal where instructors from many schools visit to find opportunities to collaborate. Drafting that proposal document is in itself a collaborative activity between instructors and other staff in schools whose goal is to have a well defined, interesting, useful and viable proposal. In more detail this requires tools to support the joint editing of documents (with support for disconnected mode and resolution of conflicts among concurrent modifications), the organization of multiple documents to prepare the activity, the definition of tasks, responsabilities, calendars, and the actual configuration of the virtual organization where the activity will take place.

When the activity has been sufficiently defined, it will be eventually approved or accepted and published in a public place (e.g., the intranet of the iEARN web site where member schools look for project proposals). Instructors from schools who have participanted or not in drafting the proposal will join. This requires the school to join the virtual organization associated to that activity, and therefore accept the rules of participation which may imply the school has to assign some people and resources or services to that project or activity.

Andrew, a student, is enrolled in this virtual project. He receives the key to the shared workspace that the instructors have created as a meeting point for the participants. This workspace already contains a list of folders describing each a number of sub-activities, the planned workflow, and timetable. Each sub-activity should be performed by two or more students in collaboration. The instructors have uploaded some PDF documents that serve as a starting point for further investigations and created a shared agenda with the events, lessons, milestones and deadlines for that project. In addition, there is a document containing a template for the final report to be prepared by each group. The students may create specific workspaces for coordination and communication within their group, either asynchronously (discussions) and synchronously (meetings), and they are also encouraged to upload their drafts or intermediary results, and to work together in editing the final report. The main workspace is for communication between all participants of the seminar, and is a common repository for information and links of general interest.

Applications should support Andrew and the instructor by enabling them to:

- Define (collaboratively) the project or activity template and instantiate a virtual organization where the activity will take place.
- Create a shared workspace or folder for a group of participants.
- Organize and view the shared calendar for the project, the group, and all the participants.
- Synchronize local changes among documents, calendars, etc., with support for resolution of conflicts.
- Prepare and perform experiments (using the resources available for this project).

In a virtual project where the specific goal is to learn collaboratively about network protocols, the following sequence occurs:

1. The educator provides learners with a document containing a number of questions concerning some protocols that can be answered after carefully observing the behaviour of such protocols in different network scenarios.
2. Individually, students devise the network scenarios that could be simulated in order to be able toanswer the questionnaire.
3. In pairs, students discuss the scenarios that they have devised individually and agree on a set of common scenarios that could be simulated in order to answer the questionnaire.

4. In groups of four, students discuss on the scenarios agreed in pairs and agree on a new set of common scenarios that will be simulated in order to answer the questionnaire.

5. Each group agrees on a distribution of the network scenarios to be simulated so that each student is responsible for creating the corresponding simulation scripts.

6. Individually, students create the simulation script that they have been assigned. Once finished, the students upload the scripts to a shared repository.

7. Each group downloads the simulation scripts created by their partners. The correctness of all the scripts is then discussed. In case it is necessary, the scripts may be modified by any of the students. The modified scripts are also uploaded to the shared repository.

8. Each group agrees on a script to be simulated. The simulation can be launched by any of the students. A trace file generated by the simulator is stored in the shared repository.

9. Each group uses the trace file to collaborative visualize an animation of the simulation and discuss it in regards of the questions posed by the educator. The visualization can be started by any student. During visualization, any student can also stop, rewind, advance or resume the animation. Besides, students can make annotations in the animation that can be seen by their partners.

10. Groups repeat steps 8 and 9, until all simulations have been visualized and discussed by the students.

11. Each group agrees on the answers to questions posed by the educator.

12. Each group writes a report of their experiments based on the traces and graphs available on a group file repository. For this they would need a collaborative text editor that support disconnected mode and resolution of conflicts. Different groups are not allowed to access each other's report.

At the end of the activity the instructor will read and evaluate the outcomes of the activity based on the final report or presentation produced by each group under their supervision. Instructors may decide to discuss among them on the overall outcomes of the project and perhaps share some of the most significant student's reports and finally write a joint report on the experience of running this project and arrange a meeting (real or virtual) to share experiences and learn from it. That information could be published on the web site of the network of schools (e.g., the iEARN web site), the instructors could archive the activity (download an archive with the relevant bits and pieces of the activity for just archival or evaluation purposes) and the virtual organization created specifically for that project will be eventually dismantled.

3 Architectural Principles

This section explains the principles behind the Grid4All architectural design. Grid4All uses a component framework, which provides autonomic management mechanisms, and is based on the concept of a virtual organisation. This section ends with a layered view of the Grid4All architecture.

3.1 Component Framework

Using components has emerged as an effective approach for building and managing complex software systems. In this approach, all system elements are constructed or wrapped as components, and management involves using primitives of the underlying component model (e.g., setting properties of components). As opposed to web service-based approaches, like WSDM, the component-based approach targets both constructing and managing systems in a unified way, thus reducing the effort for performing management tasks. Moreover, the component-based approach typically provides richer management interfaces with powerful operations, such as changing the configuration of composite components. Clearly, in applying this approach, the choice of component model is crucial. We have adopted a component model that extends Fractal, a general, reflective component model intended for building dynamically reconfigurable systems. Fractal and our extensions are examined next.

Fractal components are runtime entities that communicate exclusively through well-defined access points, called *interfaces*. Interfaces are connected through communication paths, called *bindings*. Fractal distinguishes primitive bindings from composite bindings, which contain a set of primitive bindings and other components. Composite bindings enable supporting arbitrary communication styles, such as media streaming or message queuing. Fractal distinguishes also primitive components from composite components, formed by hierarchically assembling other components. Hierarchical composition is a key Fractal feature that helps managing the complexity of understanding and developing component systems.

Each Fractal component is made of two parts: the membrane, which embodies control behaviour, and the content, which consists of a finite set of sub-components. The membrane is composed of several controllers. Fractal supports open reflective facilities in the sense that it allows arbitrary (including user-defined) forms of controllers. Nevertheless, Fractal does define a useful set of four basic controllers. The *attribute* controller supports configuring component properties. The *binding controller* supports binding and unbinding interfaces. The *content controller* supports listing, adding, and removing sub-components. The *life-cycle controller* supports starting and stopping the execution of a component. Finally, Fractal includes an architecture description language (ADL) for specifying configurations comprising components, their composition relationships, and their bindings.

Our component model extends Fractal in the following ways. Components can be remotely accessible entities, distributed over different machines. The model adds composite bindings that support remote method invocation and group communication, particularly useful for building decentralised, fault-tolerant applications. Group bindings enable invocations to be delivered either to all group members (one-to-all style) or to any, randomly-chosen group member (one-to-any style). Our component model also adds support for VO management abstractions and core services, described in the following sections.

3.2 Autonomic Management

The VOs in the Grid4All scenario are very dynamic, along two different dimensions. Members and resources of a VO are constantly changing; we use the term *churn* to refer to this dimension. For example, although a VO have a stable amount of fabric resources, individual computers join and leave the VO continuously, as members go off and on-line. Furthermore, over time, the total amount of available resources and members also changes; we call this *evolution*.

A VO might experience both churn and evolution. The distinction remains useful because the challenges are somewhat different. For instance consider a service consisting of a number of distributed components. To respond to churn, components residing on a resource that leaves are away to an alternative resource. However, as the VO evolves and service demand grows or shrinks, the service should reconfigure to scale appropriately.

In the democratic grid, VOs will be created, grow, shrink, and die off as the members' interest in the collaboration waxes and wanes. New applications and services will be injected into an existing VO. Some VOs will prove popular, some will not. The democratic grid evolves and changes at a high rate. As different populations of users pool and share their resources, churn rates are expected to be high.

In the past, adapting to problems would require labour-intensive manual administration. This is not feasible in the presence of high churn and evolution rates. Furthermore the users of a democratic grid are mostly non-professionals. The Grid4All scenarios require high-level abstractions and tools to enable VO adminstrators and service providers to formulate rules and policies for self-management.

Therefore *autonomic management* plays a great role in the Grid4All architecture. Autonomic computing aims to automate system adminstration and management. By taking humans out of the loop, labour costs decrease, response to problems is faster, and to availability improves.

Grid4All takes the *control loop* approach to autonomic management. High-level management policies are expressed in a high-level language. These are translated into rules for the the management runtime. The runtime monitors or *senses* the system, triggers *events*, which execute the associated rules. For instance, when a machine crashes, other interested machines receive the corresponding events and can recover.

In Section 4.1, we will see how overlay networks enable a robust platform for component management. Sections 4.2 and 4.3 present the high-level abstractions enabling VO autonomic management.

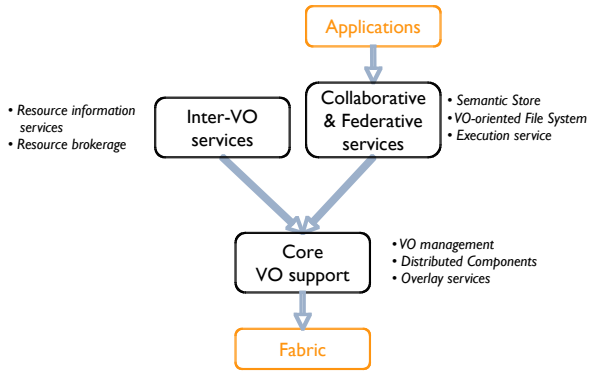


Figure 1: Major building blocks of the Grid4All architecture

3.3 Virtual Organisations

A Virtual Organisation (VO) consists of a set of resources and a set of users. Both resources and users may belong to different institutions; thus the sets may cross institutional and administrative boundaries. A VO has well-defined boundaries, which may however vary over time.

Concretely, the VO concept combines two related, but relatively orthogonal sets of mechanisms:

- Grouping and virtualising a set of resources (objects). We will call this a *virtual resource* hereafter. (A set of virtual resources is called "a grid" in OGSA.)
- Grouping and naming a set of users (subjects). We will call this a *user group* hereafter. (This is what is called "a VO" in OGSA.)

The *VO management system* (Section 4.3) and the *security infrastructure* (Section 4.4) constitute one link between these two sets of concepts. The VO management system maintains the security associations between user groups and virtual resources, as prescribed by the VO's security policy. The security infrastructure ensures that, when a user executes an operation on a resource, that operation is permitted to that user, according to the relevant policy.

The other, less standard, link between the two concepts of virtual resources and users is that the VO management system also monitors users to ensure that they fulfill their obligations as stipulated by the appropriate policy. Indeed, the placement of both users and virtual resources within the VO is motivated by the two-way interaction with the VO management system indirectly monitoring both virtual resources and users.

In general, we call the set of all users (or user groups) of a VO *members* of the VO.

3.4 Architecture of Grid4All VOs

Figure 1 summarises the major building blocks of Grid4All, their roles and their dependencies. They are as follows.

- The *fabric* includes computers, storage, files, application binaries and sources, and other basic resources provided to the VO by its members. In this document, the term "resource" without further qualification refers to a fabric resource. The fabric is assumed and not described in any detail here.
- The *Core VO Support Architecture*, described in Section 4, comprises basic support for the operation of VOs, of higher-level Grid4All services and applications: overlay services, the distributed component management service (DCMS), and VO management services.
- The *Information Services* and *Resource Brokering* (Section 5) provide facilities to find resources (outside of VOs) and to facilitate building of VOs. This comprises a match-making service, helping clients find the resources that they need, and a virtual resource market place where providers and consumers can meet and trade grid resources.

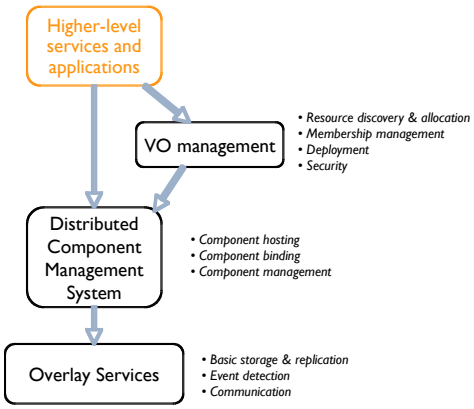


Figure 2: Architecture of Core VO support

- Our *support for collaborative and federative VOs* (Section 6) focuses on data services for sharing storage and content, i.e., a VO-oriented file system VOFS and a semantic consistency middleware Telex, and an Execution Service for deploying computation.
- *Applications* use services provided by these building blocks, e.g., to obtain computation, storage and content resources or to achieve self-* behaviour. Applications are described in other deliverables. In this document we will use only simple illustrations, centered around the network simulation scenario of Section 2.3.2.

It should be noted that for pragmatic reasons the current implementation of Grid4All does not always conform to this architecture. However it is our intent to converge all our major developments to it.

4 Core VO Support Architecture

We now present in more detail the Core VO Support sub-system. It provides basic self-* mechanisms for the operation of VOs, for higher-level Grid4All services, and for applications. Figure 2 illustrates its architecture. It comprises the following components:

- *Overlay Services* deal with connectivity of VO resources and elements. Participating nodes (i.e., computers) form a logical network called an overlay. We use structured overlay networks because of their inherent self-management properties and scalability. These services provides basic naming, communication, and data storage services that are used by upper layer services.
- The *Distributed Component Management Service (DCMS)* supports *distributed VO-aware self-** services. DCMS allows the application delevoper to program self-* functionality of *component-based services*. In what follows, we refer to the latter as *self-* code*. DCMS is overlay-based, and it delegates resource discovery and allocation to VO management services introduced next.
- The *VO Management Services* that include *Resource Discovery* and *Allocation*, *Membership Management*, *Application Deployment* and *Security* services. VO Management Services maintain information about VO users, and based on that information decide on sharing of VO resources between VO users. It also provides the basic mechanism to start new services in the VO. VO Management Services are overlay-based.

4.1 Overlay Services

The Overlay Services leverage the self-* properties of the Niche structured P2P overlay network [2], which demonstrates excellent self-organization and scalability properties. VO elements (such as components, services and applications) can leverage these properties to adapt to changing network conditions and remain available through name-based binding and communication.

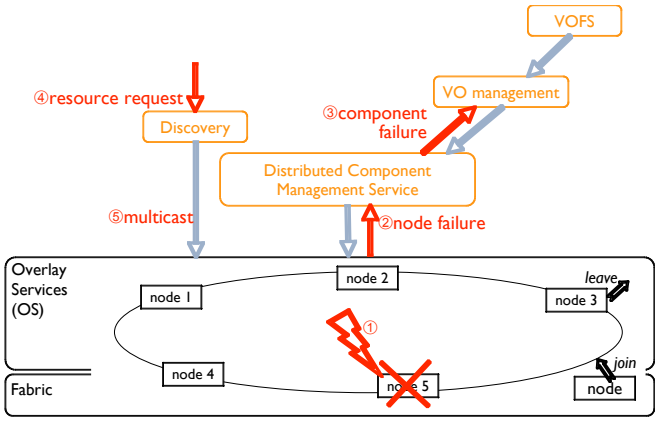


Figure 3: Overlay Services execution scenario

The set of overlay services include:

- Distributed Hash Tables (DHTs) that stores data with a specified degree of replication, despite unplanned node arrival and departure (called *churn*).
- Self-maintaining, P2P routing and communication despite churn.
- Reliable publish/subscribe service for event dissemination.
- Low-level resource discovery for VO management.
- Group communication: multicast communication service and response aggregation; name-based bindings.

The following sections describe these services in more detail.

Overlay services support building specific sensor-actuator (SA) event-driven control loops. Components across the network can react to events, for instance to an important resource becoming unavailable. The overlay services are themselves self-managing, and the SA-services they provide are robust.

Overlay services are available at every Grid4All node and serve to build the Grid4All VO management services (Section 4.3), the distributed component management framework (Section 3.1) and data management (Section 6.1). The overlay services can be directly used by applications.

4.1.1 Overlay services scenario

Figure 3 illustrates the two main functionalities of Overlay Services. The first comprises basic DHT functionality and support for the DCMS (described in a later section): (1) A failure occurs, which causes Node 5 to crash. (2) The Overlay Services detect the failure, and use the redundant capabilities of (logically) neighbouring nodes to recover OS functionality. For instance, hash table keys previously maintained by Node 5 are now the responsibility of Nodes 4 and 6. (3) Failure notification is forwarded to the DCMS, which may send a more abstract event to interested parties (in this example, the VO management layer).

The second functionality illustrates basic message routing: (4) For instance, a client sends a resource discovery request to the appropriate service. (5) The Discovery service uses OS multicasting to locate the resource.

The OS layer has basic self-management capabilities, illustrated (on the right of the figure) by fabric nodes continuously joining and leaving.

4.1.2 Publish-Subscribe Service for Event Dissemination

This service provides a reliable distributed event dissemination mechanism. It is used by management elements: *sensors*, *watchers* and *aggregators* and *managers*. Clients may subscribe to specific events occurring in Grid4All resources, components and applications, in order to react and adapt to those events. The service delivers a specific event to the clients that subscribe to the corresponding event type.

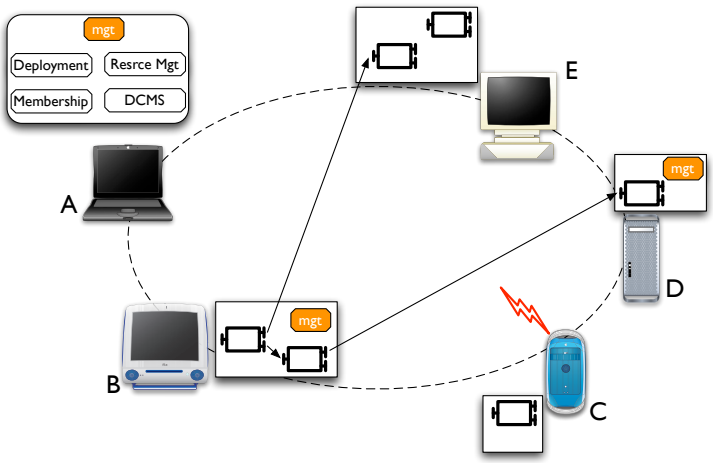


Figure 4: DCMS lifecycle: autonomic response

A sensor continuously monitors changes in a VO resource or component, and reports changes via *events* to watchers. Via appropriate sensors, a watcher monitors the status of individual component or groups of components. An aggregator subscribes to several watchers, and maintains partial information about multiple components at a coarser grain. Changes to fabric-layer resources, i.e., resources joining, leaving or failing, are continuously monitored by the VO Resource Discovery Service.

4.1.3 Low-Level Resource Discovery

The low-level resource discovery service is used by the VO management services to discover computational or storage resources given a set of low-level resource requirements, e.g., CPU speed or memory size.

4.1.4 Services for Name-Based Component Binding and Component Grouping

These services are used by the distributed management component framework. These services allow forming a group of components to be treated as a single entity. Component grouping allows the component framework to implement additional types of bindings not specified in the original Fractal model, namely name-based bindings and one-to-any and one-to-many bindings. See Section 3.1 for some details on name-based bindings.

4.1.5 Overlay services within Grid4All and innovation

The Grid4All overlay services provide novel communication and information storage and aggregation abstractions that hide dynamism of VO members and resources. Our overlay services leverage the self-* properties of the underlying structured overlay network. Overlay Services enable the self-managing and self-healing implementation of the Distributed Component Management and VO Management Services.

4.2 Distributed Component Management Service (DCMS)

Grid4All provides a programming framework and a set of related services for developing and executing distributed component-based services that run on VO resources. The framework supports design and execution of services that require minimal manual administration in dynamic VOs. The programming framework leverages the state-of-the component programming models, and the services are built on overlay networks leveraging their robustness, self-management and scalability. The central element of the programming framework is the *Distributed Component Management Service* (DCMS) that hosts components of a service and supports self-management of that service.

Figure 4 illustrates an example scenario of the DCMS. Machine B at the bottom left is dependent on two remote components, one located at E, and one that was hosted in Machine C, which has now crashed

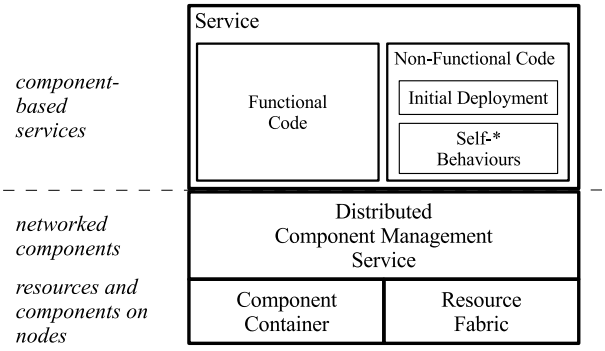


Figure 5: Grid4All Service Programming Framework

(it is greyed out in the figure). The DCMS has created an instance of the management framework on a Machine D, which has created a replacement component locally. Then the management instance at B redirects the channel, from the old component at C to the new one at D.

4.2.1 Service Programming Framework

A service code in the Grid4All framework consists of a component-based implementation of the service's functional specification, and a separate part implementing the service's non-functional requirements, i.e., *initial deployment* and *self-* behaviors*, see Figure 5. Services are executed using the Distributed Component Management Service that can be seen as a distributed virtual machine that executes service's non-functional code. Service components are first-class entities in the DCMS that are manipulated by that service's non-functional code. DCMS provides network transparency for service's functional code, and network awareness for the service's self-* code. DCMS relies on component containers that reside on VO computers, and on the VO resource fabric that provides for resources needed for component execution. DCMS delegates the resource allocation requests issued by services to the VO Management Services as described next.

4.2.2 Service Management Stack

The self-management of services in the Grid4All framework is supported by the VO Resource Discovery and Allocation Services and the *Overlay Sensing and Actuation Services* which are a part of the DCMS. The VO Management Services allow the service's non-functional code to allocate resources and monitor their availability in the VO. The overlay services allow the self-* code to monitor and manipulate the service's architecture. The self-* code makes its decisions based on service load, availability of resources and maybe direct user commands. The overlay services are implemented on top of overlay networking infrastructure, and leverage its robustness, self-management and scalability.

The *Initial Deployment* code, see Figure 5, performs initial acquisition of resources and deploys service components. The Initial Deployment code can be either manually written by the service developer, or generated by the VO management from the service's ADL description, as discussed in Section 4.3. The reflected service architecture is passed over to code implementing self-* behaviours that is organized as a distributed network of *management elements* that is executed by DCMS. At the moment, the self-* code is written by the service developer, though we envision that in some cases it can also be automatically derived from an extended ADL service specification.

4.2.3 DCMS innovation

The DCMS supports a novel programming model for designing distributed self-* Grid applications. The model separates the component-based functional part of the application from its non-functional part, which includes both initial deployment and self-* behaviours. Providing self-* behaviours is relatively straightforward thanks to the compact API of DCMS.

The DCMS implementation leverages the self-* properties of the underlying structured overlay network.

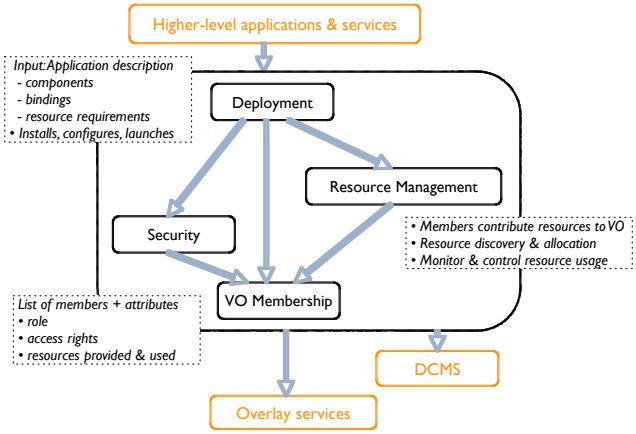


Figure 6: Architecture of VO management

4.3 VO management

As explained in Section 3.3, a VO is an abstraction that virtualises resources and groups users within a virtual administrative domain. The VO management system provides the VO abstraction and manages VO elements, namely, members, resources, and components. The VO management system addresses setting up and dissolving VOs, keeping track of VO membership, deploying application components, discovering and allocating resources, and monitoring and controlling the resource usage of applications.[2]

VO management is decomposed into four core services: membership, resource management, deployment, and security. The services expose interfaces that conform to the Fractal-based distributed component framework and build on the overlay services described in Section 4.1. The upcoming sections describe the VO management services in more detail.

4.3.1 Membership

At the core of the VO management system, the VO membership service maintains a database of VO members and associated information, including the roles and associated policies of a member, his current on-line status, and the resources he currently owns or uses. We distinguish two basic types of users (modelled as roles): standard members and administrators. Administrators can create/dissolve VOs, add and remove members, retrieve information on members, create roles and associated policies, assign roles to members, and create invitations for members. Policies constrain the VO management operations that roles can perform (e.g., a role can allocate up to a specified amount of resources). Members can register with a VO using invitations, log in, log out, and retrieve information on other members. The membership service includes support for publishing events indicating changes in the maintained information (e.g., a new member was added). The service is accessed through Fractal interfaces, but a graphical user interface is also provided.

This service also monitors users to ensure that they meet any obligations that were agreed. A typical obligation, might be that a user has his home machine on-line, available to the VO for its use, a certain number of hours per day.

4.3.2 Resource management

The resource management service enables members to contribute resources to the VO, and to discover and allocate VO resources. In the context of this service, a resource is a share of computation and storage capacity that supports component deployment; a component is deployed on a single resource. Contributing resources involves configuring and adding to the VO the local Grid4All container, or a remote container. Resource discovery relies on specifying requirements on resource properties; e.g., specifying a minimum CPU speed or memory size. Resource allocation allows reserving discovered resources or parts

[2] The security framework, described later (Section 4.4), ensures isolation between VOs and proper access to VO resources by VO members.

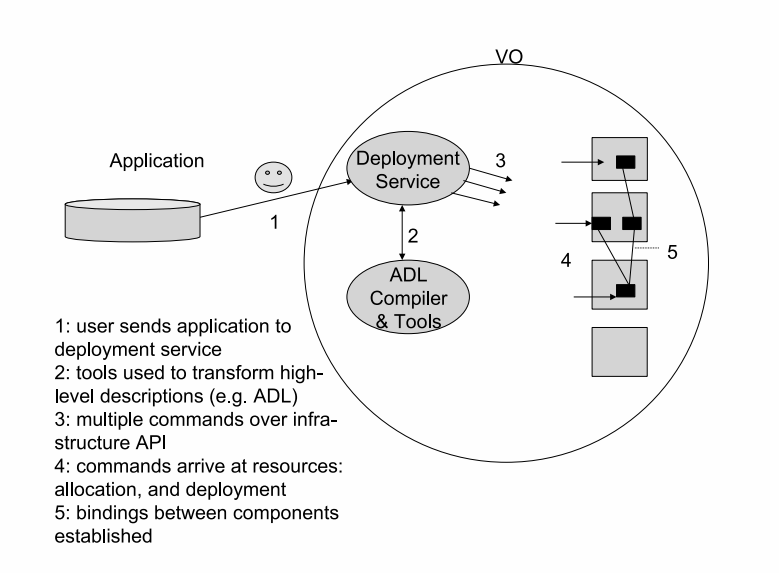


Figure 7: Overview of the deployment system

thereof. Reserving a part of a resource creates a new resource with capacity deducted from the original resource. The service includes support for publishing resource discovery events (e.g., a new resource has joined) and resource parameter change events (e.g., the CPU load of a specific resource has changed). As mentioned earlier, all supported operations are subject to the policies maintained by the membership service.

The resource management service virtualized resources as seen by service management. This allows the VO to regulate the resource usage between services prioritizing between service as determined by policy upon research scarcity.

4.3.3 Deployment

The deployment service supports installing, configuring, and activating distributed, component-based applications within the VO. The input to the service is a description of the application together with a set of requirements (i.e., the set of underlying resources). The service chooses a concrete environment on which the application will run, and performs instantiating, binding and configuring the necessary components, then executes the application.

Figure 7 illustrates the deployment service.

The configuration/deployment description includes the following information: the application architecture in terms of components, their composition relationships, and their bindings; component configurations in terms of attribute values; packaging information (e.g., the names of software packages with the code); and resource requirements for components. The description is written in an extension of the Fractal ADL. The target environment contains the hardware platform available to the VO. The package repository contains the software bundles that are to be installed, packaged according to the OSGi specification.

The deployment engine implements the deployment service. The engine is a set of Fractal components, consisting of a loader, a compiler and a back-end. The loader is responsible for verifying the correctness of the deployment description. The compiler creates tasks that are executed by the back-end. The backend provides deployment primitives that build on the distributed component framework (e.g., creating and binding components) as well as VO resource discovery and allocation service.

A final step in the deployment, after appropriate resources have been found and allocated, components deployed, configured and bound, the deployment service sets up the management elements that implements the self-* behavior of the service (see section 4.2).

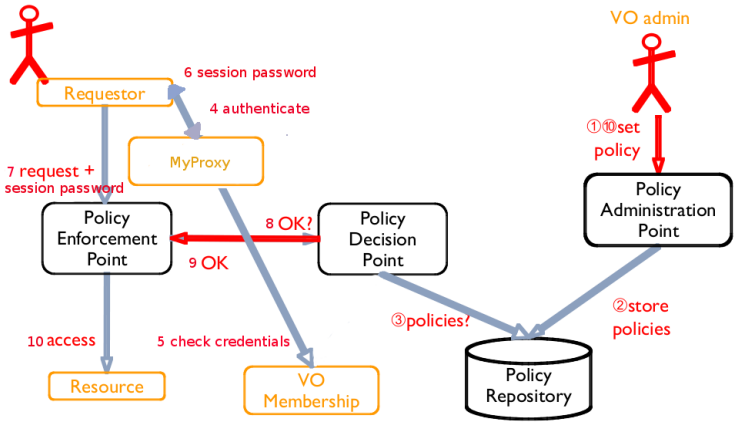


Figure 8: Architecture and interactions of security infrastructure

4.4 Security infrastructure

Security mechanisms ensure that users cannot exceed their rights. The Grid4All security infrastructure, and its interactions with other building blocks, are described in Figure 8. We focus on inter-service security, i.e., restrictions to be maintained on user invocation of a given service, or between service and service. Before that we say a few words about intra-service security. A Grid4All service generally has a (or several) front ends where users or other services may invoke them. In addition the service is composed of components residing on different machines, some of which may communicate each other as determined by the bindings in the configuration. Clearly there are issues of eavesdropping and isolation. This can be deal with by encryption and virtual machine technology. Note that the service and VO management are priviliged, only they can use the management interface of a given component.

Grid4All includes a basic policy-based security infrastructure. A VO sets a security policy, dictating how its users may access VO virtual resources. The Grid4All security infrastructure protects resources, enforcing the VO policy.

A VO member's access rights are defined by the user's *role*. A role is a dynamic identity assigned to a user or a user group, based on conditions such as group membership, or time of day.

A *policy* is a set of formal rules that define the rights of given roles to access given virtual resources.

4.4.1 Scenario

We describe a typical scenario of the interactions between the security framework and the other components of a Grid4All system. We distinguish four different phases: setting security policies, lookup, logon, and enforcement.

Setting (1) The policy administrator for a VO designs a policy through the Policy Administration Point (PAP) user interface. (2) The PAP stores the policy in a policy repository.

Lookup (3) A Policy Decision Point (PDP) looks up a set of relevant policies in the repository and caches them.

Logon (4) A user sends credentials to MyProxy. (5) My proxy checks the credentials with the VO member database. (6) MyProxy returns a session password identifying the user.

Enforcement A user interacts with some application. (7) The application requests access to a resource on behalf of the user, passing his session password as an argument. The resource is protected behind a Policy Enforcement Point (PEP). (8) The PEP sends the session password to the PDP in the "OK?" invocation. The PDP evaluates the cached policies and the cached list of users. (9) If access is authorised, it returns OK. (10) Access proceeds to the resource.

We point out several items of interest in this scenario.

- The four phases occur in order, but may be separated by arbitrary amounts of time.
- Both the PDP and the PEP cache information in order to decrease latency. In order to keep the caches consistent, the Policy Repository calls back onto the PDP, which calls back onto the PEP, when policies change. Similarly the VO membership database calls back when membership changes.
- Latency is further improved if, as in the scenario above, the PDP looks up information in advance. However this needs to be balanced against other performance measures such as memory size and maintaining consistency with the databases. Alternatively, the PDP may look up information on demand, i.e., in response to the "OK?" invocation.

4.4.2 Approach

Grid4All authentication is based on attribute certificates issued for VO members by the VO Management System (described in Section 4.3). A certificate includes the VO's identity and the user's role(s). These are used to evaluate the capabilities of the user when making an authorization decision regarding some action that the user wants to perform on some virtual resource.

The Grid4All authorization infrastructure enforces security policies to all to VO members. A resource owner can also define a local security policy. Regardless, a request to access a resource is always a consistent combination of the VO decision and the resource-owner decision: access is granted only if some policy evaluates to "permit" and no policy evaluates to "deny". Enforcement of VO security policies is obligatory.

Security policies are written in the eXtensible Access Control Markup Language (XACML).[3] Grid4All uses the ability of XACML to specify temporal conditions in policy rules. For example, access can be granted for specified days of the week only, before or after a specified time or date only, or for a specified duration only. This ability allows a resource owner to set timing constraints on usage of her resource by the VO. Policy-based authorization with time constraints are a novel approach to protecting VO resources in general, and data resources in particular.

4.4.3 Security Components

The policy-based security infrastructure in the Grid4All architecture is formed of the following security components depicted in Figure 8.

- Every sensitive resource is protected by a Policy Enforcement Point (PEP). A PEP acts as a request filter. When a program requests access to the virtual resource, the PEP communicates with its PDP, to obtain an authorization decision ("permit", "deny" or "not applicable"). The access proceeds only if the decision is "permit;" otherwise access is denied.
- A Policy Decision Point (PDP) evaluates requests in the context of a security policy. A PDP receives authorization requests from PEPs. It retrieves the appropriate policy from the Policy Repository, evaluates the request, makes the authorization decision, and returns it to the PEP.
- A Policy Repository (PR) is a persistent store for policies. Policies are expressed in the XACML language.
- A Policy Administration Point (PAP) is a user interface front end for creating security policies and for storing them in a PR. The PAP facilitates creation of XACML policies by providing a resource-specific API for setting access rights (e.g., ACLs in VO file system).
- MyProxy handles the user's certificates. This component relieves the user from the burden of getting a certificate from a well-known certificate authority, CA, and issuing temporary certificates (proxy certificates) as proof of identity. Instead MyProxy has its own CA and keeps all certificates locally. The identifier of the user is instead a session password.

[3] Our prototype is based on the open-source XACML implementation from Sun Microsystems [18].

Of course, both PAPs and PRs are protected by a PEP.

In order to hide latency, a PEP maintains a cache of recent authorization decisions. Similarly, a PDP maintains a policy cache. When the PR is updated, it invalidates upstream PDP caches. This in turn invalidates upstream PEP caches.

The security infrastructure can be used to protect any Grid4All virtual resource or service. For instance, the VOFS file system (Section 6.1) places a PEP in front of file access.

In addition to the default PEPs, a resource owner or application may place more PEPs at appropriate points. For instance the Shared Calendar application (Section 6.2.5) which distinguishes individual calendar entries inside a calendar file, protects each calendar entry access with a PEP.

4.4.4 Security in the educational scenario

Authorization is performed whenever a virtual resource is being accessed via Grid4All services in the use scenario. Teachers and students will be given different roles, with different access rights to course resources. These may include course material, simulation engines, results of education experiments, exam scores, etc. A teacher will have the right to construct and to modify access control policies associated with different course resources. The teacher will assign a student the right to access the material related to the classes in which the student is enrolled, during specific hours.

The policies must respect the privacy concerns of students. For example, exam marks of a student should be confidential, i.e. the marks should not be seen by other students unless the target student permits to view her marks. Different access operations can be performed by different roles with different purposes. To respect privacy, an access control decisions also depend on the purpose of access. An authorization decision may also be attached to obligations (e.g., to send email to a student or a teacher) to be performed by PEPs. A PEP enforces mandatory checks whenever a student wants to access the course documents (e.g., reports) or resources, e.g., simulation engine.

4.5 VO management and other Grid4All components

The membership service uses the DHT for storing membership information. The resource management service builds on the low-level resource discovery and event services; it specialises these services to handle computation and storage capabilities, and enhances them with support for policy enforcement. The deployment service builds on the low-level deployment service, adding a declarative, ADL-based interface.

The VO management services are closely coupled to the distributed component framework. All services expose Fractal interfaces conforming to the distributed component framework. Moreover, the deployment service uses the component creation and component binding primitives supported by the framework. On the other hand, the distributed component framework uses the VO resource management service to support deployment within the VO environment, conforming to VO policies.

The core VO management services are essential for forming and taking advantage of VOs. Creating VOs, adding members, and deploying components on allocated resources are relevant to all Grid4ll scenarios.

4.5.1 VO management within Grid4All and innovation

The VO management functionality is analogous to existing grid systems. However, thanks to decentralised, overlay-based implementation, Grid4All VO management services operate reliably in dynamic environments. Implementation of Grid4All VO management services is based on DCMS. This enables deployment and operation of self-managing grid applications that conform to VO policies.

5 Discovery and market services

A prelude to the formation or the growth of a VO is the discovery and acquisition of relevant resources. This section describes the corresponding mechanisms.

5.1 Information services

The Semantic Information System (SIS) provides a matching and selection service between peers that offer or request resources and services within grid environments. This service will be used by the Grid4All market place (GRIMP). In the market place, resource/service consumers and providers negotiate traded entities in auction based markets, where these markets are spontaneously initiated (instantiated) by different actors, such as resource/services providers, consumers, or 3rd party actors. Markets are accessed as services that are themselves advertised at the Semantic Information System. SIS acts as a services' registry for the following types of services: Markets, Application Services and Services exposing resources. The SIS may be queried by software agents as well as by human users to select advertised services and resources: Matchmaking happens through different criteria concerning resources and other application-specific traded domain entities, as well as services' profiles. The returned query results are ranked according to resources/services matching characteristics and providers'/consumers' features.

Objectives

- The development of a service discovery service using ontologies, allowing semantic and syntactic interoperability. The semantic approach for discovery is currently proven to be a key enabler for the facilitating the construction of automated discovery of services within an open environment.
- Semantic matchmaking of annotated WSDL specifications of services to OWL-S profiles.
- Open, extensible, and interoperable mechanism
- Personalization of selection based on user preferences and reputation

Approach SIS exploits the Grid4All resources ontology, as well as OWL-S services profile specifications. The system matches resources descriptions and services' specifications at the semantic (in opposition to the syntactic or lexical) level. Particularly, for resource-exposition services, the matching of resources requested-offered is performed at the semantic level, using the ontology for grid4all resources and services. Furthermore, other types of market related, application-oriented and offers/requests related properties can be exploited for matchmaking.

Entities (either human or software) pose queries to SIS. These queries are either requests (seeking for matching offers) or offers (seeking for matching requests). The matchmaking component match queries to corresponding entries in the registry and ranks results according to their similarity, as well as according to providers/buyers features.

SIS comprises the following components:

- **Matchmaking component:** This component matches requests and offers. Matchmaking is based on the grid4all ontology for the description of traded resources using subsumption and querying/ filtering techniques. Specifically, matchmaking of services is performed by exploiting service profile specifications in OWL-S. The output of this component is a ranked list of end point references of markets or other services that have been requested/offered. This component will be used by trading agents such as buyers and sellers. The component will also be used by the portal interface component.
- **Selection component:** This reduces the set of relevant consumers/providers, found by the Matchmaking Component, 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 trading such a resource), the providers are those sites that have such a file (respectively the running markets themselves). In fact, the selection component takes into account the satisfaction of providers/consumers given the results they get from SIS.

Services and functionalities

- **Advertising Service:** Discoverable services are advertised by submitting their WSDL description and additional annotations. The advertising service provides an abstraction of the underlying ontology so as entities using the service should not be aware of this at a low-level implementation .
- **Querying Service:** Queries are submitted as OWL-S descriptions of requested or offered services that are matched against available advertised services. Matching is based on subsumption and querying using appropriate query languages such as SPARQL.

Interfaces Advertising and querying services of SIS have both a web service and a web interface. A web-based user interface is used for the insertion of data for advertised services and service annotations, as well as for performing queries. The objective here is also to render the ontology description transparent to the user. Requests and offers inserted through the web forms are inserted in the SIS registry. The elements of the user interface are automatically generated based on the ontology schema. A web service interface accessible through SOAP will also be available.

SIS and the educational scenario SIS supports search and selection of markets that trade in resources matching requirements set forth by consumers. Markets may also be selected based on their specific properties, allowing consumers to restrict their search to the most relevant market.

Any third party-initiated market can advertise itself in the SIS so as to be discovered by provisional buyers and sellers. Application developers from WP4 can query the SIS for discovering application and storage services.

SIS innovation Whereas to current approaches use a single reference ontology, the SIS performs match-making using different, heterogeneous domain ontologies. This technique permits multiple annotations of a WSDL element, sourced from different ontologies. Furthermore, our annotations are stored in an external semantic annotation file, which should facilitate adoption and exploitation by industry.

5.2 Resource brokerage services

Resources model computational and storage elements needed to execute applications. Brokering concerns arbitration and allocation of Grid resources to virtual organisations. VOs lease resources on need by negotiating at the resource market place.

Grid4All model of virtual organisations are similar to peer-to-peer networked applications. Members of VOs use resources that belong to the VO. In Grid4All we allow VOs to incorporate resources from non-members. Such resources are expected to be leased from resource brokers that select and match of consumers and suppliers.

Objectives

When there are fluctuations in supply and demand we need mechanisms to arbitrate whose requests for resources should be satisfied and by who. Priority based, proportional sharing allocations are mechanisms that are useful when the different consumers (applications or users requiring resources) belong to the same organisation; here consumers may be prioritized or allocated shares. This is not the case when applications and users from multiple independent VOs contend for resources. Market based brokering with pricing mechanisms provide fair arbitration, gives incentives and is decentralized.

The minimal GRIMP (Grid4All Market Place) addresses:

- Selection of suitable resources,
- Decentralized feedback from market to aid traders in their negotiation,
- Mechanisms to allocate and establish prices,
- Protocol to establish agreements between consumers and provider.

Approach The resource brokering problem is decomposed into multiple replaceable and extensible components allowing different use cases to use a subset of these services to realize their objectives. The platform provides tools that allow traders (consumers and providers) and mediating agents to create ad-hoc auction markets, where they can negotiate for resources. Divergence of prices and hot-spots are risks of market places with simultaneous auction markets. GRIMP provides a decentralized publish/subscribe based information service that allows trading agents to react to market situations by publishing aggregated information of the market place. The minimal platform consists of tool-kits and basic services and may be extended to provide added-value higher level brokerage and mediating services. The market-place itself will use the VO run-time services for deployment, monitoring, and other self-management.

5.2.1 Services and functionalities

This section describes the core functionality.

Auction servers Its service is accessed through common Market Interface. Participants send *bids* specifying the resources, quantities, time constraints and budget and auction determines allocations and the transaction prices. Behind a unique facet (interface), auction servers may implement different mechanisms (rules and policies for bidding, price and allocation determination). The server is architected as a set of Fractal components that separate economic and system concerns. The server is designed as a tool-kit within which specific allocation and pricing policies and rules may be plugged in. Auction servers are advertised at the Semantic Information System which may be queried by traders to obtain candidate lists of active auction markets.

Within this framework two proof-of-concept economic mechanisms have been implemented.

- *K-pricing double auction*: This mechanism is used to acquire storage resources to satisfy the scenario described in Section 2. This mechanism is computationally efficient when trading a type of resource amongst multiple consumers and providers.
- *Combinatorial auction*: This mechanism allows allocation of bundles of computational and storage resources.

Market factory This is a repository service provided by the market place and offers two sets of interfaces:

- To designers and developers to register implementations for a new type of auction mechanism,
- A selection mechanism for traders (consumers and providers) by which these can choose a specific auction format and request instantiation of a new auction server.

Market Information Service In a decentralized market place where there are multiple auctions that execute simultaneously and where there a large number of participants, it is important to obtain synthetic and summarized information; aggregated information such as average prices, average demand. Consider a VO executing an application on compute nodes mainly from one geographical region (Paris). This application may prefer to allocate additional resources in the same region but may decide to compromise this preference if current prices of close resource are higher than its valuation for the resources.

Currency Management Service This is a decentralized account and banking service providing basic functions to create accounts and transfer funds between accounts. Technically it implements a transactional mechanism on top of a DHT to implement decentralized accounts.

5.2.2 Properties

The GRIMP architecture presents the following non-functional capabilities to address the requirements.

Scability and reliability The federated architecture allows the market place to scale to thousands of users.

- Auction servers are created spontaneously on need and mechanisms with appropriate properties (computational complexity, efficiency) can be chosen on demand.
- The Market Information Service (MIS) allows participants to be informed of market situation including aggregated and summary information and thus react to market situation.

Configurability In an open environment the type of applications, the type of resources needed cannot be fixed. Support for multiple auction mechanisms and tools to select appropriate mechanisms renders the market place configurable. As an example, an application requiring both computational and storage resources requires combinatorial auctions, where as for an application requiring only a single type of resource (computations or storage) auction mechanisms capable of trading single items is sufficient.

The architecture based on the Fractal component model presents the advantages are a single coherent solution to address design, deployment, configuration, and assembly.

5.3 Discussion

5.3.1 Discovery and market services in the educational scenario

Our scenario (Section 2) involves a classroom where student groups execute a network simulation program that requires sizable computational and storage resources. The ability to allocate resources on demand to the VO enables instructors to flexibly organize their classes without high upfront investments.

The educational scenario also requires acquiring storage resources to maintain the project content. These storage resources may be acquired from the market, in order to increase the capacity of the VO's shared workspace (see Section 6.1). It illustrates the need for multiple mechanism. Indeed, the first case requires a combinatorial auction, whereas the second one requires only K-pricing double auction, which is less demanding computationally.

5.3.2 Discovery and market services within Grid4All

The market place is itself implemented using the VO run-time so as to use the core services such as deployment (of auction servers on demand), to monitor nodes and components, to execute the components of the MIS and the Currency Management System. This is a technical usage of the VO management run-time to implement the services of the market place.

The Market Information and the Currency Management services are implemented as a decentralized peer-to-peer applications. They use the overlay services for communication, persistence, and event notification.

Traders (consumers and providers) select auction servers by querying the SIS where active markets advertise themselves.

5.3.3 Innovation

- Separation of execution management from resource brokering and resource allocation,
- Decentralization through co-habitation of multiple simultaneous auction services and flow of information through the MIS,
- Fractal-based architecture for configurable auction servers,
- Selection of auction formats through the market factory,
- Specification of interaction protocols using WS-Choreography and generation of auction process workflow in BPEL.

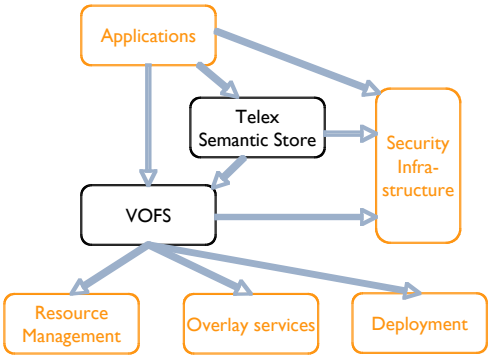


Figure 9: Architecture of Grid4All data services

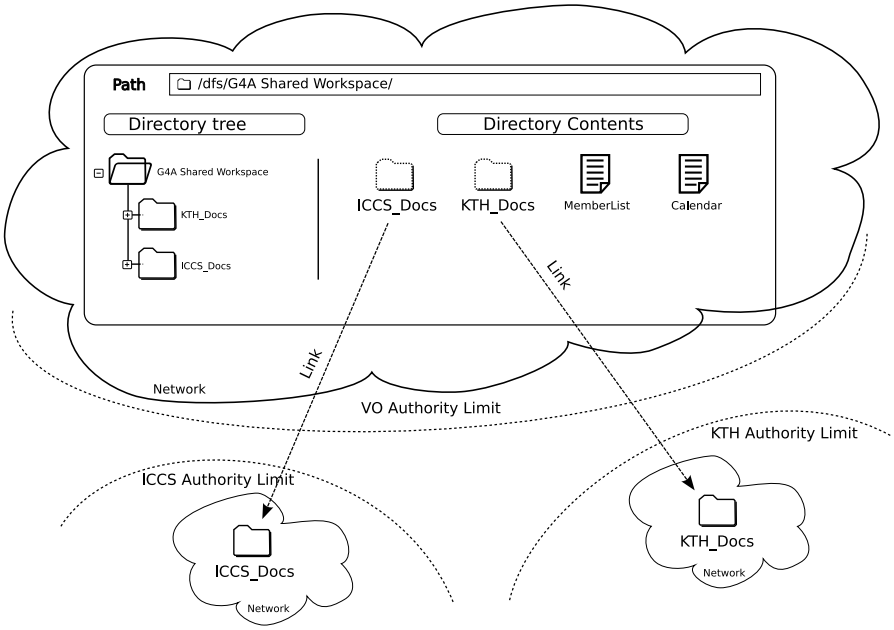


Figure 10: VOFS shared workspace: The VO administrator of G4A has created a shared workspace where members ICCS and KTH safely export documents which they continue to own and completely control.

6 Support for collaborative and federative VOs

The Democratic Grid is designed to be used mainly in a collaborative fashion, within VOs formed of resources federated from its members. It should make it easy to share information and to work on it together, but must nonetheless be secure and maintain isolation. This section describes the enabling mechanisms for collaborative and federative VOs. We focus in particular on data sharing and execution management. Figure 9 illustrates the architecture of the Grid4All data services.

6.1 Federative VO workspaces and VO-oriented file system

Grid4All features a novel VO-oriented file system, called VOFS, designed for sharing and collaboration within VOs. VO members may pool storage resources that they own, as well as resources acquired from external providers.

A VO is materialised as a *shared workspace* that federates the files and directories exposed by the VO users. Furthermore, VOFS supports the Telex semantic store, described in Section 6.2.

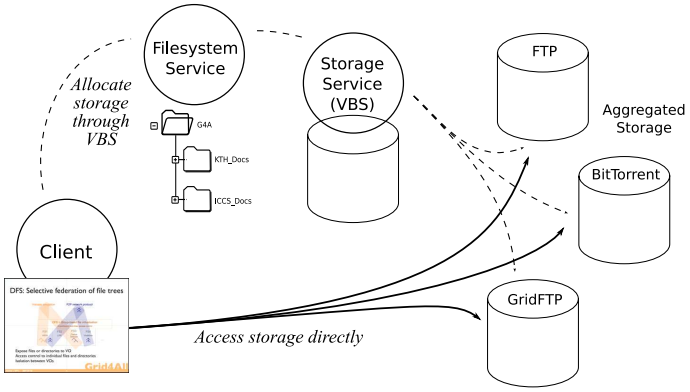


Figure 11: Allocation and access of filesystem storage.

6.1.1 The VOFS network

The VOFS is a network of peers. Each peer may possess data resources such as file namespaces, file data or raw storage. A resource owner may expose its resources to be shared by a VO, without compromising their control over them.

The identity of a peer is defined cryptographically with a public-private key pair. Peers can authenticate themselves and their communication. They can also protect their data and their communication with encryption. The owner of the identity may be a physical person, or a software entity, or a cooperating group of either (e.g., a VO, see Figure 10). Resources are protected using the the Grid4All security infrastructure as described in Section 4.4.

VOFS is a federated file system. Each peer may link foreign resources into its own resources. This makes them accessible exactly like the peer's own resources. This is somewhat similar to links in a Web page, making federated content accessible from a single place.

VOs make extensive use of federation by setting up shared workspaces. In the example setup of Figure 10, a VO named G4A is formed and ICCS and KTH join as members. The administrator creates a shared workspace for G4A, into which only he can authorise linking (and thus, federation) of members' resources. Similarly, members ICCS and KTH completely control their own documents which have been exported to the G4A Shared Workspace. In a VOFS network, aggregation and federatino of resources cannot compromise the owners' control over them.

VOFS is built on top of a lower filesystem and storage layer, VBS+DFS, which provides all the peering and resource access mechanisms. VOFS adds VO-awareness on top of VBS+DFS.

6.1.2 Resources

Our federative file system supports two types of data resource: filesystem (managed by the DFS layer) and storage (managed by the VBS layer).

DFS filesystem resources are hierarchical file namespaces including (logically) the *content* of files. Storage resources are raw disk storage. Note that these two types of resource are quite different. Storage is a fungible resource, i.e., functionally any two storage blocks are interchangeable. However, for reliability, performance or availability, the same content may be replicated into several distinct blocks. On the other hand, two distinct files are not interchangeable, but replicas of the same file are.

Resources owned by peers are globally identified with URIs constructed from the identity of their owner peers together with a local identification. Local identification can be a file path or an arbitrary string. These URIs are the references used in peer-to-peer links, or any other resource designation.

Returning to the web page analogy, owners can shape their (peers') resources as freely as one can edit their webpage. Owners can also link to foreign resources, but as in the web, they cannot control the resources they link to. This is an important consideration for the architecture (see Figure 10). This is particularly convenient for creating VO-wide shared workspaces where everyone's resources are available for reading while owners only write their own.

Access to a resource is requested with a well-defined communication *request token*. Request tokens

always include identification for both the *authority*, who owns the resource to be accessed, and the *agent* who requests the access, providing accountability throughout the network. Authorities have different communication endpoints for each type of resource, called *services*; the destination service for a token can either be a filesystem service or storage service. A resource URI encodes the cryptographic identity of the authority, the authority service and a path or handle for local identification (for filesystem or storage resources, respectively).

6.1.3 Security

VOFS protects resources by filtering access requests through a Policy Enforcement Point (PEP) as described in the security infrastructure of Section 4.4. The PEP contacts a VO policy service to make authorisation decisions. In addition to VOFS request tokens, the PEP may also need further requestor credentials managed by the policy service itself, and not VOFS. Users acquire these credentials and then associate them with specific files in their VOFS client. VOFS forwards those credentials to the PEP along with the request tokens.

6.1.4 Working with VOFS

The basic application interface to VOFS is a locally mounted POSIX-like filesystem, implemented by the DFS layer. Applications access files with normal POSIX calls. There exist special, virtual files that allow control over the VOFS-specific functionality, again over POSIX-only calls with extended semantics.

As client peers navigate through the network, they cache remote resources. Cached resources remain persistent for disconnected operation. Also, unsuccesful attempts for remote access are persistently logged so that they can be retried after network recovery or application restart.

The VOFS provides loose, eventual, consistency semantics. Conflicting concurrent accesses will not be resolved, but can be detected.[4]

VOFS has primitives for publish-subscribe communication and remote event notification. The mechanisms are used to handle the significant asynchrony in the network, which is further amplified by disconnected operation.

Aggregation by cross-peer filesystem links logically brings remote files into a local directory. Similarly, remote storage resources can be logically joined together to a single, aggregated image, called a storage pool, which is managed by the VBS subsystem. Storage allocation for new or expanding files are made from such pools. Every filesystem service is associated with a primary storage pool. Storage can be offered to specific peers for use in their filesystems, by registering storage resources to the specific peer's primary storage pool.

Storage allocation is a distinct function from storage access. Allocation is performed by the VBS, while access is requested by the clients and is served, initially, also by the VBS. The architecture allows pluggable modules that implement allocation and access for different storage servers, such as HTTP, FTP, BitTorrent, or Grid-specific ones. As depicted in Figure 11, the dirfferent storage resources with their own access protocols are registered with a primary storage service (VBS) associated with a specific filesystem service. Clients using this service, allocate storage through the service but subsequently access storage directly.

6.1.5 VOFS innovation

Prevalent distributed filesystems, such as Coda, NFS or AFS, were not designed for today's computing environment, which is Internet-wide, dynamic, and strongly present in so many human activites. The filesystem is being displaced, in its role as the primary context of user activity, by the Web and by peer-to-peer applications. The traditional systems are slowly expanding their niche towards this kind of combination but their basic principles have not changed (e.g., NFS version 4.1).

In contrast, VOFS boldly offers connectivity and support for collaboration to both users and applications in a distributed environment, thanks to a unique combination of traditional and modern features.

[4] The Telex semantic store (Section 6.2) provides conflict resolution and consistency semantics for collaborative documents stored in VOFS.

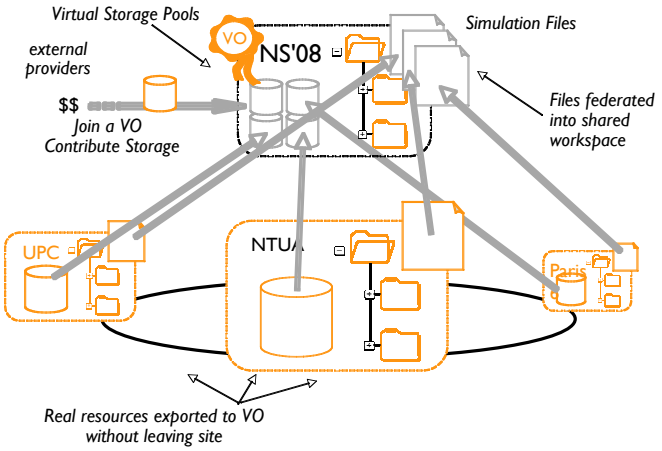


Figure 12: Network Simulation collaboration scenario: storage and file sharing

6.1.6 VOFS in the educational scenario

In an educational setting, for each participant, VOFS is a tool providing a gateway to the environment distributed storage and file resources. The users can readily expose their files, contribute content and storage to a cause, or browse and use resources over the network.

Most useful are the *shared workspaces* that can be created by linking distributed files and storage into a separately administrated filesystem, owned by a VO, an application or a user. In the Grid4All scenario, the Telex document (see Section 6.2), is one such application-controlled shared workspace, enabling the Semantic Store's powerful collaborative functionality.

Figure 12 illustrates how storage and files may be shared in the Network Simulation scenario (Section 2.3.2). Students from three universities, UPC, NTUA and Paris-6, collaborating on a particular set of simulations set up a VO called "NS'08".

Each set of users contributes some storage space to the collaboration, as indicated by the "disk" logo of each participant exposed, through the arrows, into the NS'08 virtual workspace. As the simulation files are very large, the VO acquires additional sotrae space (for a fee) with some external providers. VOFS consolidates all this remote disk space into a virtual shared disk, as indicated by the multi-disk logo in the NS'O8 workspace. This storage space, physically located at four different places, is remotely accessible by all the members of the VO as if it was local.

Now the students start the simulation and produce result files. These may be allocated, either from each students local storage, or from the virtual shared storage pool. Again, the students expose their files to all users in the VO by naming them in the shared virtual file hierarchy depicted at the top right of the NS'O8 workspace.

6.2 Semantic storage for collaborative documents

In order to collaborate, users must be able to write to the same data, be it a file, a database, or more generally a set of documents. VOFS files (Section 6.1) provide only best-effort consistency, and are therefore not sufficient for true collaboration.

The actions of different users may conflict; understanding and resolving these conflicts requires taking application semantics into account. The Semantic Store of the Grid4All architecture, called Telex, is a common middleware enabling collaborative work modes for any application.

6.2.1 Telex Semantic Store

The Telex middleware enables remote sharing of documents and remote collaboration (Figure 13). Telex facilitates the design of collaborative applications by taking care of complex application-independent aspects, such as replication, conflict repair, and ensuring eventual commitment. Telex allows an application to access a local replica without synchronizing with peer sites. The application makes progress, executing

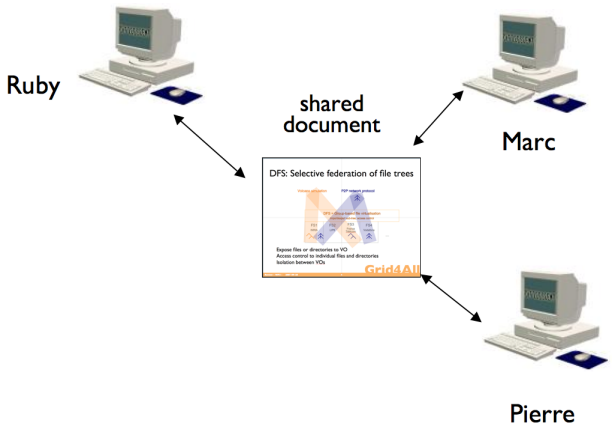


Figure 13: Collaboration over a shared document

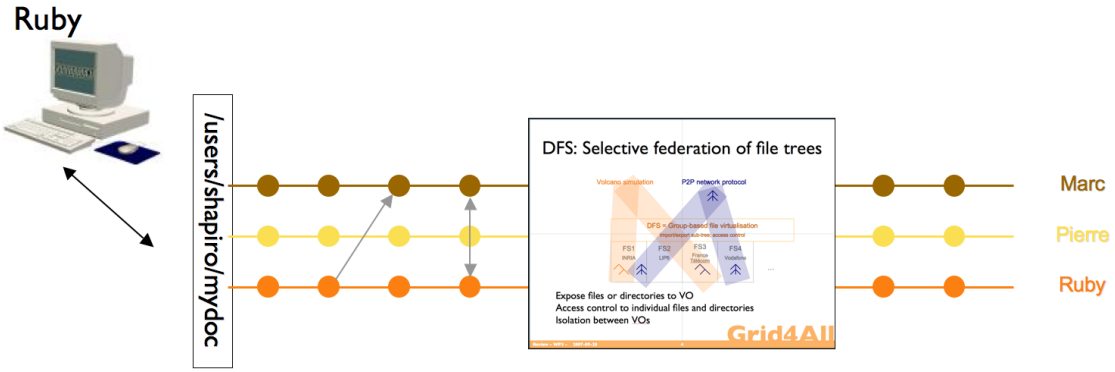


Figure 14: Structure of a Telex document stored in VOFS

uncommitted operations, even while peers are disconnected. Telex supports the whole spectrum of collaboration modes, from on-line sharing ("what you see is what I see") to synchrounous, off-line, check-out/check-in isolated work.

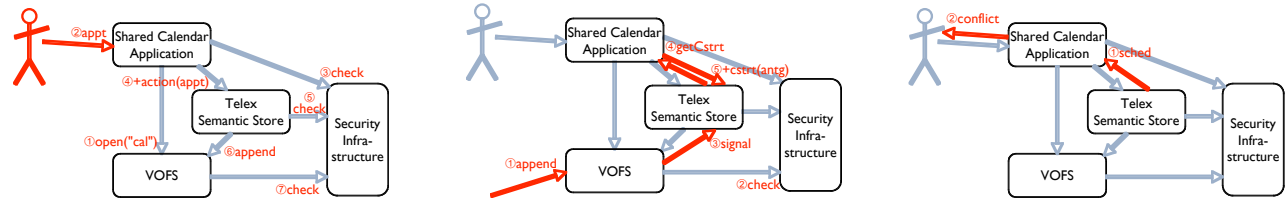
Telex implements the Action-Constraint Framework (ACF), a novel approach to collaborative data sharing, based on a replicated and persistent graph [16]. The nodes, called actions, represent application events, such as some user calling some command. The vertices, called constraints, represent specific semantic relations between actions, such as causal dependence or conflict. Both actions and constraints are supplied by applications. Telex records the graph persistently, makes it available in a decentralised environment, compares possible outcomes of conflict resolution, and selects and commits one of them.

The ACF approach allows sites to work independently if required, but also to remain aware of concurrent events and their consequences. Telex allows separation of concerns: the middleware performs the common but difficult tasks of collaborative applications; application developers can focus on semantics. As applications share the same engine, novel cross-application scenarios are possible.

6.2.2 Accessing and storing a collaborative document

Telex integrates seamlessly into Grid4All's VOFS (Section 6.1). A collaborative document is persistent, can be named just like ordinary files, and is accessible by all members of any VO it is exposed to. Access to collaborative documents is subject to access control through the security infrastructure (Section 4.4).

A document is stored in VOFS as a special directory type, containing a log file per user. An application writes the actions and constraints of a given user into the corresponding log file. VOFS ensures that its contents is persistent and can be accessed by all authorised users. As there is no concurrent writing, consistency is not a problem. As different documents have distinct logs, the system should have good locality.



a. App. reifies user op. as actions & constraints b. Remote action rcvd.: upcall for conflict constraints c. Compute schedule(s), execute, display

Figure 15: Telex interactions

Furthermore, the separate log files can each have different access control policies, which increases security. Telex will perform fine-grain security checks, by calling a PEP to validate each action and constraint (this is in addition to the coarse-grain security at the VOFS level).

Figure 14 illustrates an example. User Marc has created a shared document (a slide about VOFS) and has exposed it under the name /users/shapiro/mydoc in the VO name space. Users Marc, Ruby and Pierre may together modify the document, hence the three logs. The arrows (constraints) indicate that Marc's third action depends on Ruby's second update, and that Marc's fourth action conflicts with Ruby's fourth one. The picture shows Ruby interacting with her replica; Marc and Pierre also have their own replicas. The picture also shows the state of the document, as seen by Ruby, after applying four actions from each of Marc, Pierre, and Ruby. The other users may be viewing a different state.

6.2.3 Telex Interactions

Figure 15 illustrates the interactions between Telex and the rest of the Grid4All system: the application, the security framework, and VOFS. We distinguish three phases: local interaction, remote interaction, and execution. In this example, a user is interacting with a Shared Calendar (SC) application (Section 6.2.5).

Phase a. Local interaction: processing an application event in Telex.

(1) When the user connects, the SC application opens the corresponding ”cal” document in VOFS (actually this is mediated by Telex).

(2) The user performs some action, for instance creating a new appointment.

(3) The application checks with the security framework if the user is authorised to create this particular appointment. Let's assume the check is successful.

(4) The application sends the “create appointment” action and any constraints to Telex.

(5) Telex checks if SC is allowed to create these entries on behalf of this user. Note that this check is not redundant with check (3).

(6) Telex writes the entries for the action and the constraints in the user's log file in the ”cal” document.

(7) VOFS checks that Telex is allowed to write to this log file on behalf of SC on behalf of this user. Again, this check is not redundant with (3) or (5).

Phase b. Remote interaction: processing events received from remote Telex sites.

(1) The VOFS daemon receives a remote write appending to some log file.

(2) VOFS checks whether this write is legal. Note that this check is not redundant, since in general it is not known whether the remote site is trusted. Assume the check succeeds.

(3) VOFS notifies Telex that the log has changed.

(4) Telex reads the new entries, and discovers an action that might conflict with an SC action this site already knows. Telex upcalls the application to find out if this is a real conflict.

(5) Assume this is the case. The application responds by logging the corresponding conflict constraint (an antagonism in this example).

Phase c. Scheduling: sending events to the application.

(1) When Telex has received a sufficient number of actions and constraints, it computes one or more new schedules (according to parameters set by the application).

(2) This particular schedule contains a conflict. The SC application displays the conflict for the user.

Note that the three phases are completely asynchronous.

6.2.4 Telex innovation

In contrast to previous systems such as Coda, Bayou or Unison, Telex is based on a principled framework (ACF). Application designers focus on semantics and need not worry about distribution or conflict resolution. Hence, writing a collaborative application is expected to be much easier with Telex.

Previous implementations of ACF were proof-of-concept prototypes [15, 12, 1]. Telex is the first complete implementation, and is planned to be thoroughly tested and evaluated under realistic applications. Telex is designed for robustness, flexibility and performance, in a VO context. Furthermore, the Telex commitment protocol is decentralised and does not rely on a central site (in contrast to previous work, such as Bayou).

6.2.5 Telex in the educational scenario

Telex support for replication and reconciliation is helpful for any kind of collaborative application. Remote classroom activities, such as a CVS-like tool for co-operative editing, or a decision support exercise that examines a number of "what-if" scenarios, could be developed above Telex. Specifically, the Grid4All project is developing two applications above Telex, the Collaborative Filing System (CFS) and the Shared Calendar (SC). CFS is a basic remote collaboration tool, whereby VO-authorised users may manage a library of files and file versions; CFS uses Telex to manage updates to the library and to detect and resolve conflicts. SC is a simple prototype of a joint decision-making application. Authorised users manage their calendars collaboratively. One user may create and manage a meeting, e.g., changing the time or the list of attendees; this may cause conflicts with other meetings, which are detected by Telex; Telex proposes possible resolutions and users vote for their preference; finally Telex identifies and commits the winning resolution.

6.3 Execution management

Execution Management addresses issues of scheduling, instantiating and managing to completion of *work units* within a virtual organization. An Execution Management is needed for sharing computing resources among a VO and executing applications on other VO nodes. For example, in the class network simulation scenario, multiple tasks are created for parallelizing the work to be done where each task corresponds to the simulation script started by student groups. Then some nodes will be allocated for those tasks and execution will be monitored to enforce this scheduling.

6.3.1 Objectives

Execution management addresses the following concerns:

- Management of the set of resources that at a time belong to a virtual organization,
- Finding and selection of resource that match requestor criteria,
- Provisioning or reservation of selected resources at required times to applications.
- Placement or scheduling of work units,
- Deployment or installation of executables
- Life-cycle management

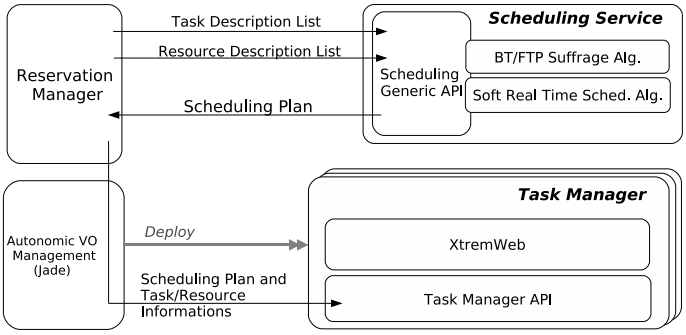


Figure 16: Overview of Execution Management

6.3.2 Approach

Current Grid computing systems propose a high-level service referred to as the Job Manager that is typically implemented as queuing systems. It encapsulates jobs and their management where job may range from a simple task to complex workflows. Thus job managers basically arbitrate allocation of computational and storage resources to multiple applications submitted by multiple users.

Grid4All focuses on application specific execution management leveraging on the core VO management services such as deployment, monitoring, and discovery. Higher level work load management that arbitrates between multiple users and multiple applications are not our focus, but may either be integrated or implemented using the core VO management facilities.

The main abstractions relevant to execution management are:

- Tasks: units of work; they are represented as components that implement or wrap the corresponding logic (e.g., the UNIX executable) and act as management points.
- Resources: hosting environments for tasks; they are represented as Grid4All containers.

6.3.3 Services and functionalities

The core execution management services rely on the availability of other management services as described in Section 4.3.

Scheduling service The scheduler is an off-line planner for bags of independent tasks that builds schedules, that is, placements of tasks onto resources taking into account time constraints. Its inputs are the set of task descriptions and the set of available resources and their characteristics. It does the matching according to a BitTorrent/FTP [17] suffrage algorithm or a soft real time (deadline) scheduling algorithm. It outputs a plan describing when and on which resource each task is to be executed. The scheduler may be used either a library or can be used to implement a planning service.

The inputs to the schedule planner are:

- **Task description list**: A task is specified by the executable file, the size of input and output data and the scheduling related parameters, i.e., earliest starting time, latest ending time, and worst case execution time.
- **A resource description list**: A resource is specified by its availability (start and end time) and its performance parameters (CPU speed, memory size, available network bandwidth). Alternatively the scheduler also accepts the real resource list, i.e., the addresses of the compute nodes that will be used to execute the different tasks.

The objective function of the scheduling heuristics is to minimize the make-span, i.e., to minimize the time to execute the total set of tasks of an application. The planner incorporates two algorithms that are accessed through a common generic interface:

- The BT/FTP/ suffrage algorithm [17], an enhancement of the suffrage algorithm [6] that accounts P2P data transfer costs in wide area networks.

- A novel planner for soft real-time applications such as simulators (network simulation for example), multi-media/video conversion application.

The scheduler chooses an appropriate algorithm and outputs an execution plan that is for each resource in the resource list, provides the sequence of tasks that should be executed on that resource.

Reservation manager Grid4All virtual organizations are dynamic and may need to negotiate for and acquire new resources from resource brokers such as the market place. The reservation manager (or buyer agent), hides the complexity of negotiating at the market place and provides a simple interface to applications, managers, and higher-level services executing within a VO.

Task manager This encapsulates the aspects of real execution of schedule plans on the set of physical resources. The task manager interacts with core VO services (principally deployment and life-cycle management to install executables and monitor their termination. The input to the task manager consists of:

- A set of Grid4All containers,
- Tasks packaged as deployable components,
- The work plan itself, that is, the order of execution of each of the tasks on each of the referenced Grid4All containers.

The implementation of the task manager is based on XtremWeb [3], a well-known desk-top grid computing system which is itself a monolithic implementation of a complete job management system. The task manager itself is deployed using the core services and it in turn deploys application specific tasks. The task manager manages the execution and correct termination of the tasks and terminates when the input bag of tasks is emptied.

6.3.4 Interfaces

The scheduler can be used either as a library by an application that needs an execution plan for its set of tasks, or may be rendered available to all applications of the VO as a service. The common interface hides the specific algorithms that are implemented. The scheduler chooses the best algorithm transparently to the invoker.

The task manager may be either deployed as a service or used by a specific application manager to execute the work plan, that is, to manage the deployment and life-cycle management of the tasks of the application. The output of tasks will be written back to the final destination place as has been specified within the task description.

6.3.5 Execution management and the educational scenario

The use scenario described in Section 2 requires that the different student groups execute the same network simulation software. The schedule planner will be used to propose a plan of execution of the different instances of the simulator on resources that have been previously leased by the reservation service. Execution of multiple copies of the same software (with different input scripts) is assimilated to execution of a bag of tasks. The objective of the class room is to minimize the time of execution of the total set of simulations. The BT suffrage algorithm incorporates both transfer of data and also the differences in bandwidth available to each of the compute nodes that execute the set of simulations.

6.3.6 Execution management within Grid4All

The reservation manager uses VO management services to discover and allocate resources. Given that tasks are represented as components, the task manager uses the deployment service to instantiate tasks, and the management interfaces (defined in the component framework) to monitor and control tasks dynamically.

The reservation manager may lease resources from the market place, adding them to the VO.

7 Conclusion

This paper presented the motivating scenario for Grid4All and the Grid4All architectural vision. The educational scenario is a reference for both functional and non-functional requirements of the technical work-packages.

Grid4All enables a wide range of society with access to grids. This includes both professionally-operated grids, and the capability for end-users to build collaborative virtual organisations by federating their own resources. For this to happen, usability, flexibility and manageability must be increased, and complexity and operating costs in the presence of dynamic change must be reduced, with respect to existing grid systems such as Globus. Hence our decision to build Grid4All upon the Niche peer-to-peer technology and the Fractal component framework. Furthermore we propose innovative collaborative and federative VO functionality. Finally, we propose novel discovery and market services to flexibly extend the resource pool accessible to a VO.

We described the overall system architecture and the set of software packages designed for Grid4All. We provided a high-level view of each software component, its capabilities, its relevance to the educational scenario, and its position within the architecture. Technological detail described in other deliverables are not repeated here.

At the lowest level, overlay services support a distributed component management service framework, which hides the complexity of the underlying fabric. The framework enables self-managing applications and system services. Developers of higher layer functionality may focus on the functional concerns, and management complexity to the component framework. the Grid4All scenarios.

Grid4All provides support for novel forms of collaboration. The VO-aware file system VOFS allows VO members to pool storage resources and to add storage acquired from outside sources, possibly using the Grid4All market mechanisms. VOFS also enables VO members to share designated file hierarchy and content in a VO-specific work space. The Telex semantic consistency enables collaborative applications such as consistent file sharing and authoring, or complex decision-making applications. Finally Grid4All provides a VO-oriented deployment and execution service.

Two other companion documents are relevant: Deliverable D 4.7 presents some key measurable requirements, and Deliverable D 5.4 presents our evaluation criteria and test plan. Measurable requirements and their evaluation will consolidate the claims to innovation as presented within this deliverable. A later document will illustrate the deployment architecture and some prototypical use cases.

References

[1] Reza Akbarinia, Vidal Martins, Esther Pacitti, and Patrick Valduriez. Design and implementation of Atlas P2P architecture. In R. Baldoni, G. Cortese, and F. Davide, editors, *Global Data Management*. IOS Press, 2006.

[2] P. Brand, J. Hoglund, K. Popov, N. de Palma, F. Boyer, N. Parlvanzas, V.Vlassov, and A. Al-Shishtawy. The role of overlay services in a self-managing framework for dynamic virtual organizations. In *CoreGRID W. on Grid Prog. Models, Grid and P2P Sys. Arch., Grid Sys., Tools and Environments*, Heraklion, Crete, Greece, 2007. To appear in Springer CoreGRID series.

[3] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky. Computing on large scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Gen. Comp. Sc. (FGCS)*, 2004.

[4] J.M. Carroll. Five reasons for scenario-based design. *Interacting with Computers*, 13(1):43–60, 2000.

[5] J.M. Carroll and M.B. Rosson. Getting around the task-artifact cycle: how to make claims and design by scenario. *Trans. on Info. Sys. (TOIS)*, 10(2):181–212, 1992.

[6] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Heterogeneous Comp. W. (HCW)*, Cancun, Mexico, 2000.

[7] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, version 1.5. OGF Grid Final Document GFD-I.080, Open Grid Forum, September 2006.

[8] Welcome to grid'5000 web site. Web site http://www.planet-lab.org/.

[9] International educational and research network (iEARN). http://www.iearn.org.

[10] M. Jarke, X.T. Bui, and J.M. Carroll. Scenario management: An interdisciplinary approach. *Requirements Engineering*, 3(3):155–173, 1998.

[11] D.W. Johnson, R.T. Johnson, and K.A. Smith. *Active Learning: Cooperation in the College Classroom*. Interaction Book Co., 1998.

[12] James O'Brien and Marc Shapiro. An application framework for nomadic, collaborative applications. In *Int. Conf. on Dist. App. and Interop. Sys. (DAIS)*, pages 48–63, Bologna, Italy, June 2006. IFIP WG 6.1.

[13] One laptop per child (OLPC). Web site http://www.laptop.org/.

[14] PlanetLab, an open platform for developing, deploying, and accessing planetary-scale services. Web site http://www.planet-lab.org/.

[15] Nuno Preguiça, Marc Shapiro, and Caroline Matheson. Semantics-based reconciliation for collaborative and mobile environments. In *Int. Conf. on Coop. Info. Sys. (CoopIS)*, volume 2888 of *Lecture Notes in Comp. Sc.*, pages 38–55, Catania, Sicily, Italy, November 2003. Springer-Verlag GmbH.

[16] Marc Shapiro, Karthikeyan Bhargavan, and Nishith Krishna. A constraint-based formalism for consistency in replicated systems. In *Int. Conf. on Principles of Dist. Sys. (OPODIS)*, number 3544 in Lecture Notes in Comp. Sc., pages 331–345, Grenoble, France, December 2004.

[17] Baohua Wei, Gilles Fedak, and Franck Cappello. Scheduling independent tasks sharing large data distributed with BitTorrent. In *Grid'2005 Workshop*, Seattle, Washington, USA, 2005.

[18] Sun's XACML implementation. Web site http://sunxacml.sourceforge.net/.

Level of confidentiality and dissemination

By default, each document created within Grid4All is © Grid4All Consortium Members and should be considered confidential. Corresponding legal mentions are included in the document templates and should not be removed, unless a more restricted copyright applies (e.g. at subproject level, organisation level etc.).

In the Grid4All Description of Work (DoW), and in the future yearly updates of the 18-months implementation plan, all deliverables listed in Section 7.7 have a specific dissemination level. This dissemination level shall be mentioned in the document (a specific section for this is included in the template, both on the cover page and in the footer of each page).

The dissemination level can be defined for each document using one of the following codes:

PU = Public.
PP = Restricted to other programme participants (including the EC services).
RE = Restricted to a group specified by the Consortium (including the EC services).
CO = Confidential, only for members of the Consortium (including the EC services).
INT = Internal, only for members of the Consortium (excluding the EC services).

This level typically applies to internal working documents, meeting minutes etc., and cannot be used for contractual project deliverables.

It is possible to create later a public version of (part of) a restricted document, under the condition that the owners of the restricted document agree collectively in writing to release this public version. In this case, a new document code should be given so as to distinguish between the different versions.