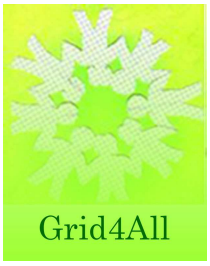

Project no. 034567

Grid4All

Specific Targeted Research Project (STREP)

Thematic Priority 2: Information Society Technologies

D4.2: Specification of situations derived from applications

Due date of deliverable: 1st June 2007

Actual submission date: 28th June 2007

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

Organisation name of lead contractor for this deliverable: UPC

Contributors: Gabriel Belvedere, Antares; Lamia Benmouffok, INRIA-LIP6; Sergio Borgoñoz, Antares; Miguel Bote, UPC; Alícia Bou, Antares; Eduardo Gómez, UPC; Joan Manuel Marquès, UPC; Leandro Navarro, UPC; Miguel Valero, UPC; Xavier Vilajosana, UPC.

Release 1

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

Table of Contents

Table of figures

Abbreviations used in this document

Abbreviation / acronym	Description
ACF	Action-Constraint Framework
API	Application Programming Interface
BPEL4WS	Business Process Language for Web Services
CBSE	Component Based Software Engineering
CFS	Collaborative File Sharing
CNSE	Collaborative Network Simulator Environment
DFS	Distributed File System
HTTP	HyperText Transfer Protocol
MVC	Model-View-Controller
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture
OR	Optimistic Replication
PR	Pessimistic Replication
QoS	Quality of Service
SC	Shared Calendar
SCA	Service Component Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SS	Semantic Store
UDDI	Universal Description, Discovery and Integration
VO	Virtual Organization
WP	Work Package
WS	Web Service
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
WSRP	Web Services for Remote Portlets

Grid4All list of participants

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

Executive Summary

This documents reports work done in task 4.2 regarding the definition of a programming model suitable for the development of Grid4All applications. Service Oriented Computing is discussed to this concern, and related to other Grid4All technologies. Four applications, representative of several of the scenarios described in D4.1, are described, and their infrastructure requirements detailed. Finally, a discussion on the need for selective transparency mechanisms is presented.

1. Introduction

This deliverable occurs in the context of task 4.2:

T4.2: Application design in Grid4All environment (M6-M24; INRIA, UPC, Antares, FT)**:**
- Evaluate situations specific for this environment such as concurrent operations, conflict, capturing and handling exceptions, failures and degradation, dealing with disconnected operation. Define selective transparency mechanisms to hide these situations and develop a realization compatible with the Grid4All model. This work will be reported in D4.2.
- Define and develop prototype an application API and programming models: mapping from application concepts, modules and calls to the infrastructure, may require redesigning and developing parts of existing or new applications. This work will be reported in D4.3.
- Define and develop prototype middleware for providing selective transparencies of specific situations. This work will be reported in D4.4.

As D4.1 already presents several usage scenarios of Grid4All applications, this report deepens further in some of these applications, to gain insight in the infrastructure requirements. Prior to the proposal of the applications and the description of their architecture, a suitable programming model must be chosen. Service Oriented Computing (SOC) offers several benefits, such as self description, platform and language independence, low coupling and coarse granularity, which make this distribution paradigm suitable for developing grid applications. In fact, the usage of services simplifies several issues concerning how to share and manage resources in a grid context, such as resource virtualization, discovery and aggregation [7]. This fact has made SOC the preferred choice to build grid applications nowadays, as described in the Open Grid Services Architecture (OGSA) framework [13]. The Web Services Resource Framework (WSRF) [16] is a collection of specifications that complements the widely used Web Services specifications [4] [3] in order enable the possibility of using services that meet the requirements imposed by OGSA. However, in Grid4All it is not always convenient to develop grid applications as services from scratch. Thus, the relationship between SOC and Grid4All technologies must be explored to avoid this. Fortunately, the Service Component Architecture (SCA) [1] defines a framework for developing web services made up of components. Thus, components can be wrapped as services, and they can also invoke web services interfaces, independently of their actual internal implementation. This applies very well to Fractal, which is used in WP1 to deploy the overlay infrastructure.

Using this programming model, four different applications will illustrate the possibilities of Grid4All technology:

- The eMeeting application will support users to interact in separate virtual room, using text, data, voice and video. It will illustrate synchronous interactions with shared text and data in the context of virtual organizations. It could benefit from the scheduling and reservation mechanisms being developed in Grid4All. It will also serve to test some functionalities of the infrastructure such as those related to distributed storage, conflict handling and virtual organization management.
- The Collaborative Network Simulator Environment will allow educators in the Computer Networks domain to design activities for their students that are intensive in network simulation, profiting from the computational and storage resources available in a grid. It is a complex application made up of several services, which could also be used in other applications. This way, it will illustrate the potential of the service oriented paradigm. Besides, it will use several Grid4All infrastructure mechanisms related to data management, service discovery, authentication and delegation.
- The Collaborative File Sharing application will allow users to have a unified view of a distributed file system, and to share and version files on it. To do so, it will implement optimistic resolution of conflicting operations, using a conflict solver that is expected from Grid4All infrastructure. Of course, this application will also make use of data management and authentication mechanisms from the infrastructure.
- The Shared Calendar application will allow participants to share their own agenda, invite others to meetings, or alter other people's agendas. It will allow disconnected usage and will solve concurrency problems by optimistic replication, which is more suitable for a dynamic environment such a grid. This application will make extensive usage of Grid4All infrastructure, of

components related to authentication, replication and communication, resource management, publication/subscription mechanisms…

Some of the scenarios that can be devised for using these applications do not benefit from the transparency offered by a traditional grid infrastructure, though most will. For example, when using video or audio streaming it may be convenient that users know that their distance may be affecting the quality of the service. The calendar application may benefit from knowing (and letting users know) users locations in terms of allowing or disallowing some operations, such as setting local time, triggering alarms or making specific appointments. In summary, the infrastructure should let applications select the type of transparency they wish to have. This need of selective transparency is further detailed in this report.

The rest of the document is organized as follows. Section 2 discusses the adequacy of service oriented computing as a programming paradigm for Grid4All. Section 3 describes the four proposed applications, their architecture and their specific dependencies on the infrastructure. Section 4 discusses the need for selective transparency. Finally, section 5 summarizes the main conclusions.

2. Programming model

Grid infrastructures are nowadays typically built following the Service-Oriented Computing (SOC) paradigm. This section first provides an overview of SOC and its adoption in grid computing. Then, the suitability of the service-oriented approach for the development of the applications to be run on top of Grid4All infrastructure is discussed.

2.1 Service-Oriented Computing

2.1.1 Service definition

A service is a software element, installed, configured, run and maintained by a provider organization, in order to offer its functionality to any potential user [18]. Services offered by different providers can be easily integrated by third-party providers in order to offer new *composed* services. This possibility stems from the distinctive features of services:

- **Self description** [17]. The description of a service may include:
  - The description of the functionality offered by the service. For this purpose, keywords could be used if a suitable taxonomy has been defined. Other approaches such as the use of ontologies may foster service discovery by end users.
  - The description of the service interface, consisting of the list of operations supported, and the input and output messages (call and return parameters) exchanged in each operation. Exceptions thrown by these operations may also be described here.
  - A quality of service description, consisting of a list of variable/value pairs that could be used by the scheduler in order to select the best services, in terms of functional (e.g. is delegation supported) or non-functional parameters (e.g. which is the cost of the invocation of the service, which is the average response time, or the average availability)
- **Platform and language independence** [17]. The description of a service interface does depend on neither the service platform nor the programming language used to implement the service.
- **Low coupling** [20]**.** The functionality of a service is typically decoupled from the functionality of other services.
- **Coarse granularity** [19] [3]. Services are software elements that tend to offer complex functionalities.

2.1.2 Types of services

According to [14], two types of services can be offered:

- **Data-oriented services** are the most common ones. Providers of these services do not offer any presentation logic to use them, mostly because these types of services are thought to be integrated with other services, and sometimes to be used by end users with their own clients. In the latter case, the end user (or some third-party) must develop a client to use the service.
- **Presentation-oriented services** are conceived to allow more complex interaction with users. In this case, the provider also offers a client implementing the presentation logic to use the service. These services can in turn be composed of several data- or presentation-oriented services.

2.1.3 Basic interaction model in Service-Oriented Computing

Besides the service concept, the service-oriented computing paradigm also defines a basic interaction model, which shows how a client can contact the services offering a functionality that suits its needs. This model, shown in Figure 1, includes the following interactions:

- Providers contact a directory service, a registry in which they publish the description of the services they offer.
- These directories are consulted by clients searching for services that suit specific needs (not necessarily just functional).
- Once a service is located, the client can use the description of its interface to know how to communicate with the service, and thus interact with it.

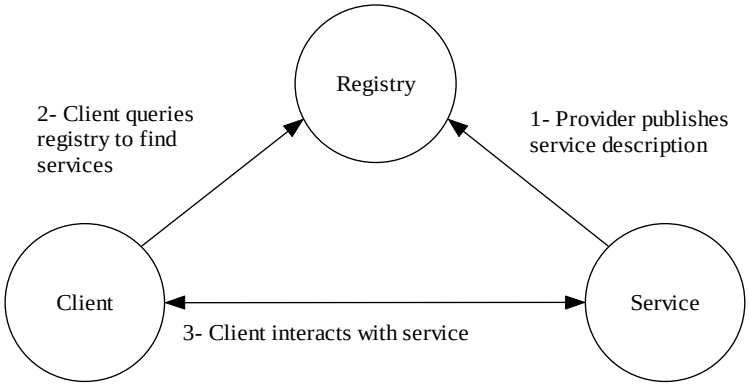

Figure 1: Basic interaction model in the service-oriented computing paradigm.

2.1.4 Relationship with Component-Based Software Engineering

Component Based Software Engineering (CBSE) [9] is a paradigm for the design, development and execution of distributed applications. In the particular case of CBSE, the basic unit to compose applications is a software *component*. The main differences between services and components are as follows:

- Services are software elements with a coarser granularity than components [19] [3] [18].
- As opposed to SOC implementations widely available, existing component model implementations have limitations in the usage of Internet-wide communications [3]. Because of this reason, components are most used to develop distributed applications that will run in a smaller environment, such as local networks or smaller intranets, although this project will provide solutions for this problem.
- Service oriented architecture states that the provider is in charge of installing, configuring, running and managing the service [18] [23]. This way, it is easier for an organization (other than the provider's) to use the functionality of the service without having to bother about these issues. On the contrary, in the component model, all these tasks must be carried out by the organization that will eventually use the components.

According to this, it can be stated that these two distributed computing paradigms are not excluding, but rather complementary. In fact, the literature shows examples of the integration of components to offer a coarser grain software element in the form of a service [11].

2.1.5 Web services

Web services [4] [3] are a particular implementation (doubtless the most popular one) of the service oriented architecture. In the web services implementation, each web service is identified by a unique Uniform Resource Identifier (URI). Further, web services use open standards for defining data formats, as well as standard protocols from the Internet suite.

Accordingly, the interaction with a web service is typically carried out through the exchange of SOAP (Simple Object Access Protocol) messages [22], transported over HTTP or TCP. SOAP messages have a very simple structure: they consist of an XML element with two parts, one for the header and other for the body of the message. The content of these parts is arbitrary, but must be structured also in XML. The general structure of a SOAP message is shown in Figure 2.

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP:Header>
  <!-- here comes the header of the message -->
 </SOAP:Header>
 <SOAP:Body>
  <!-- here commes the body of the message -->
 </SOAP:Body>
</SOAP:Envelope>
```

Figure 2: Structure of a SOAP message.

In addition, the description of the web service interface is done with WSDL (Web Service Definition Language) [21]. Again, there are two parts: an abstract definition of the service in the application level, and a concrete definition of the details that depend on the protocols that the client should use to access the service functionality. This separation facilitates the reuse of the abstract part when the same service is offered to be accessed through different protocols, or with the same one but with different communication details.

The abstract part of the interface definition defines the messages that can be exchanged, in terms of the data types carried with them. Besides, the operations exposed by the service are listed, stating the type of input and output messages for each of them. Operations are colleted in interfaces called *portTypes* in the WSDL jargon. Figure 3 shows an example of WSDL to declare the interface of a service that exposes a single operation carrying out the addition of two integers.

```
<message name="addInputMessage">
 <part name="number1" type="xsd:int"/>
 <part name="number2" type="xsd:int"/>
</message>

<message name="addOutputMessage">
 <part name="result" type="xsd:int"/>
</message>

<portType name="AdditionServicePortType">
 <operation name="add">
  <input message="tns:addInputMessage "/>
  <output message="tns:addOutputMessage "/>
 </operation>
</portType>
```

Figure 3: Example of usage of WSDL for the abstract declaration of a service interface.

The concrete part of the description specifies which protocols should be used to interact with the service (e.g. SOAP over HTTP), how are the messages being codified, and which is the URI to access the service. Figure 4 shows how WSDL is used to declare the concrete part of the addition service described above.

```
<binding name="AdditionServiceSoapBinding"
type="tns:AdditionServicePortType">
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="add">
  <soap: operationstyle="rpc"
      soapAction="http://gsic.tel.uva.es/examples/add"/>
  <input>
   <soap:body use="encoded"
      namespace="http://gsic.tel.uva.es/examples/add"
      encodingstyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <soap:body use="encoded"
      namespace="http://gsic.tel.uva.es/examples/add"
      encodingstyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
</binding>

<service name="additionservice">
 <port name="additionServicePort"
      binding="tns:AdditionServiceSoapBinding">
  <soap:address location="http://gsic.tel.uva.es/additionservice"/>
```

```
 </port>
</service>
```

Figure 4: Example of usage of WSDL for the declaration of the concrete part of a service.

Another relevant specification in the context of web services is UDDI (Universal Description, Discovery and Integration) [15]. UDDI defines several issues concerning the directories in which providers can register the web services they offer, attaching their description, so that clients can search for them. Other significant specifications are BPEL4WS (Business Process Language for Web Services) [2] and WSFL (Web Services Flow Language) [10]. Both specifications are used for service composition. Finally, WSRP (Web Services for Remote Portlets) [14] defines how the providers of presentation oriented services must distribute the clients to their services, and how these clients can be installed and used in a different context.

2.2 Service-oriented grids

The Service Oriented Architecture is currently considered the most suitable paradigm for the implementation of grids [8]. In a service-oriented grid all the resources shared in by different providers must be exposed as services, in the sense defined by SOA.

2.2.1 Motivation for the adoption of SOC in computational grids

SOA is considered a suitable approach for the implementation of the grid architecture because it simplifies several issues concerning how to share and manage resources in a grid context. More specifically, some of these issues are [7]:

- **Resource virtualization.** Resources exposed as a service offer an interface description that hides the actual resource implementation or location to the user, no matter if the resource is hardware or software.
- **Resource discovery.** SOA model eases the search of resources through the creation of registries that contain information about available resources. These registries are themselves offered as services, and can be queried by users in order to find the resources that better suit their needs.
- **Resource aggregation.** The fact that resources have a well defined interface facilitates their aggregated usage, since the client only needs to know this interface to invoke the service operations.

2.2.2 OGSA

The Open Grid Service Architecture (OGSA) [13] is a conceptual framework developed by the Open Grid Forum (OGF) [12] that defines the usage of services within the computational grid. According to OGSA, any type of resource to be shared in a service-oriented grid should be offered through a service interface that is independent of the actual resource implementation.

The OGSA framework establishes the specific requirements that must be met by services to be used in computational grids. These requirements include:

- **State representation and manipulation**. The resources typically shared in a grid are stateful. As a consequence, OGSA establishes that services must enable the possibility of accessing and managing state, and related activities.
- **Notifications**. OGSA establishes a notification mechanism between service providers and service clients that allows the asynchronous communication of changes in the state of the service.
- **Naming policy.** OGSA defines a three-level naming convention. More specifically, every named entity is associated with an (optional) human-oriented name, an abstract name, and an address.
- **Security.** Security is a very important issue in grid infrastructures. This is why OGSA states the need to enable the possibility of interacting with services in a secure way.

2.2.3 WSRF

The Web Services Resource Framework (WSRF) [16] is a collection of specifications that complements current web service specifications in order enable the possibility of using services that meet the requirements imposed by OGSA. More specifically, WSRF consists on the four following specifications:

- **WS-ResourceProperties**. This specification defines the interfaces that must be used to access, modify and query the properties that represent the state of resources.
- **WS-ResourceLifetime.** The WS-ResourceLifetime specification standardizes the way in which the logic representation of a resource may be created or destroyed.
- **WS-ServiceGroup.** This specification defines a means of representing and managing heterogeneous collections of web services.
- **WS-BaseFaults.** The WS-BaseFaults specification provides a standard way of representing faults during the invocation of a web service operation.

There are also some specifications closely related to WSRF, although they are not part of it:

- **WS-Notification.** WS-Notification is another collection of specifications that define a standard approach to notification using a topic-based publish/subscribe pattern within the context of web services.
- **WS-Addressing.** This specification provides the mechanisms required to address web services and messages.

2.3 SOA and other Grid4All technologies

As stated above, SOA is becoming more and more accepted to build grid systems. However, building systems that use SOA from scratch can be sometimes costly, and may involve re-developing applications or functional modules already available and tested. To avoid this, several efforts have been done to allow wrapping as services other modules (either legacy or new) developed using a different technology.

Among these approaches, the *Service Component Architecture* (SCA) is supported by major software vendors [1]. It aims at building applications and systems using SOA, by designing applications consisting of components that implement business logic. These components will offer their capabilities through service-oriented interfaces and will consume functions offered by other components also through service-oriented interfaces, called service references. In order to build an SCA-based application, two tasks must be completed: the **implementation** of service components, and the **assembly** of sets of components to build business applications, through the wiring of service references to services.

As shown in Figure 5, SCA supports two levels of assembly:

- Tightly coupled components may communicate through component interfaces, such as the two elements in Composite A. Inter-component communication protocols are used here.
- Loosely coupled components serve to form services that in turn can be used to build more complex services. As SCA builds upon the Web Services standard, the services formed in this way communicate through SOAP.

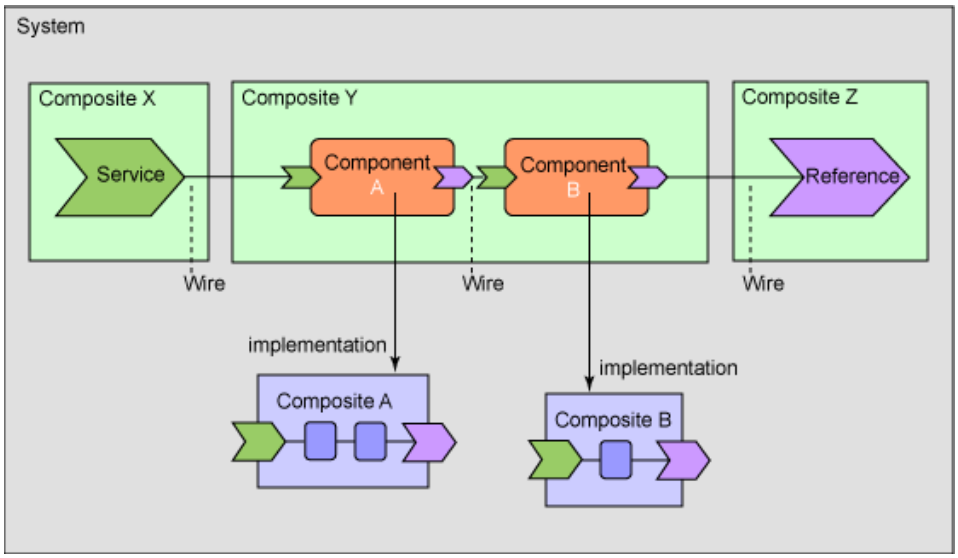

Figure 5: Compositional scheme of the Service Component Architecture [1]

Interestingly, the Fractal component model that is used in WP1 to deploy the overlay infrastructure, and that could be used to develop some of the proposed applications, interoperates well with SCA applications, because of the usage of SOAP as a common protocol, and because Fractal allows different levels of binding. In order to allow Fractal applications to invoke Web Services, the WSDL interface of these web services must be Java-annotated, and from it a new Fractal component, called *SCA Shadow*, will be generated [5]. Equivalently, the Fractal ADL can be used to generate a WSDL file of the exposed interface, and register it into the service container [5]. Further, if Proactive (one of Fractal's implementations) is used, Fractal components can be instructed programmatically to generate a WSDL interface that allows to expose themselves as web services [6].

Overall, this offers several benefits:

- Reuse of existing or development of new Fractal applications that can be easily integrated with SOA based applications, with few and decoupled additional elements.
- The code for these elements can be generated automatically from the interfaces.
- These interfaces are provided in the standard way. Fractal ADL is used to describe the application architecture, and the communication between components, and of WSDL files to describe its exterior interfaces, or to describe the services invoked by Fractal applications.
- The self-management, self-configuration or self-healing properties of Fractal component-based applications can be exploited to build more adaptive and robust services.

3. Applications and API

This section presents four applications that illustrate some of the scenarios described in D4.1, and their dependences on the infrastructure API. The choice of applications has been influenced by the experience and expertise of the partners, the relation with the scenarios in D4.1, the interest of the application domain and the potential to exercise and expose the features of the work in other work packages. The eMeeting application will support users to interact in separate virtual room, using text, voice and video. The Collaborative Network Simulator Environment will allow educators in the Computer Networks domain to design activities for their students that are intensive in network simulation, profiting from the computational and storage resources available in a grid. The Collaborative File Sharing application will allow users to have a unified view of a distributed file system, and to share and version files on it. Finally, the Shared Calendar application will allow participants to share their own agenda, invite others to meetings, or alter other people's agendas, supporting disconnected operation.

These applications depend on the infrastructure in different degrees. All relay on distributed storage and virtual organization support provided by the infrastructure. Besides, some make use of communication, resource management and allocation and conflict/resolution mechanisms. The specific needs of each application are discussed in each of the following subsections.

3.1 The eMeeting application

The main focus on the grid is to facilitate the communication between applications but the communication between people is also an important requirement, because before a group of people can use a tool they may need to agree upon how to use it. So a tool that facilitates and encourages the human-human communication is essential.

This application addresses the problem of separation, making feasible to recreate communication methods with synchronous interaction between people by replacing physical presence with virtual presence with some advantages provided by supporting tools included in this application.

The eMeeting application gives response to scenarios "Live tutorized online sessions" and "Live sessions using collaborative tools" set on sections 5.1.5 and 5.2.2 of Deliverable 4.1, allowing situations in which the weight of communication is only in one direction (one or few senders to multiple receivers) or shared by all those who take part (multiple receivers and multiple senders).

3.1.1 Description of the application

The eMeeting application is formed by two components: the administration system and the front-end application.

The administration will be used by some administrator users designated by a, for instance, VO manager, and then they can manage user permissions so end users can be added to specific rooms, or allow them to join or make their own rooms for working together.

The front end application will allow a user to join a room and start sharing video, audio and data with other users being part of the same room.

The eMeeting application needs a communication service to stream the audio, video and data generated from one user to the others connected to the same instance of the application. So, it must be able to connect to it from an application server and also from the application running on the client side. This brings the need of a service that provides the connection method and the URL to access the service in a specific location.

For the profiles handling, this application needs an Authentication system to specify who can use the back end or the front end. Those profiles will be managed by the application administration system, and must be stored in a particular database that must be provided in the configuration phase.

Once all the users are connected they can start sharing information, so the application will access a shared distributed file system from time to time for uploading slides for making presentations, save polls they generate on the fly, or to save the application logs.

3.1.2 Infrastructure requirements

The eMeeting application will need the following features from the infrastructure:

- Distributed file system access. For the in and out operations regarding the storage of the polling, slides, and log files.
- Virtual Organization management. To retrieve information about rooms and participants, and grant or deny access to each user. The application will then handle the permissions required for using the admin section or the public section.
- Service discovery. This will allow the application to be discover, select and use different services or tools as part of the eMeeting environment.
- Resolution of potentially conflicting dating and/or calling for a meeting. The eMeeting will need a scheduling application in order to date events and call users for a concrete event. The conflict solver expected on Grid4All infrastructure will help avoiding concurrent uses of rooms and also duplicated users (in the same event or being called for different events at the same time).

3.1.3 Infrastructure's API requirements

- **Virtual Organization management**
  - Obtain list of participants, roles, user credentials
- **Distributed file system**
  - Upload/download shared file
  - Create/remove/modify/list Files
  - Create/remove/modify/list Directories
- **Conflict handling**
  - Apply change
  - Resolve conflict
- **Service discovery**
  - Find and lookup services

3.2 Collaborative Network Simulator Environment

The Collaborative Network Simulator Environment (CNSE) application can be used to support the scenario "collaborative simulation of computer networks" described in section 4.1.3 of Deliverable 4.1. As stated there, the usage of this application within a grid context will allow overcoming several problems: first, the processing resources available will be much larger, and thus the complexity of the simulations will not be limited by this factor (this will benefit from the scheduling mechanisms developed in WP2, Task 2.4); further, storage will also be available on demand (supported by the Virtual File System developed in WP3, Task 3.1), and thus the number of simulations and visualizations can be that required by the educational objectives, and not restricted by locally available disks.

3.2.1 Description of the application

The design of any service-oriented application implies splitting up its functionality in a set of different services. This operation is typically performed trying to maximize the possibility of reusing the services to build other (different) applications while keeping their usual coarse granularity. Each of the identified services can be offered by one or more organizations participating in the grid which run them using their own local resources. The services that will be composed to offer the CNSE can be seen in Figure 6 and are described later in this section.

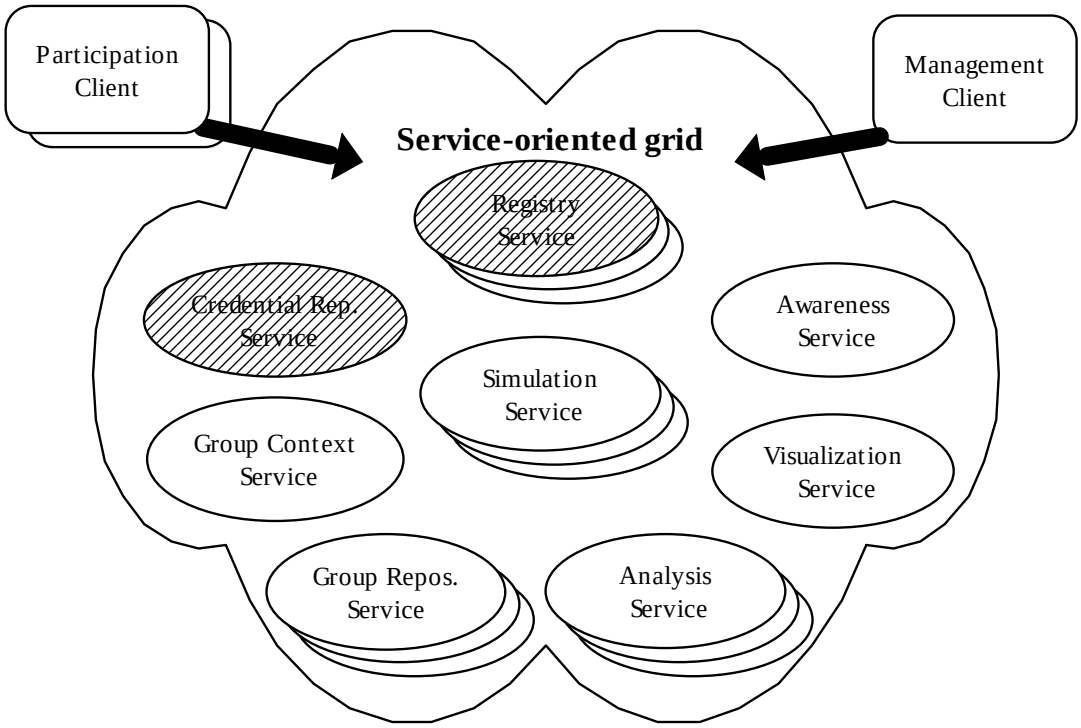

Figure 6. Services and clients of CNSE. The services expected from the infrastructure are represented with a slashed background.

Before describing these services, it must be said that CNSE will be developed reusing the well-known *ns2* network simulator and its visualization companion tool *nam*. Though these are not grid applications, their functionality is convenient to fulfil the objectives of the scenario, and thus they will be offered in the grid as services. This choice determines somehow the data flow between the different services in CNSE. At this point, it suffices to know that *ns2* must be provided with an input file, which is a script containing the description of the topology to be simulated and some relevant parameters for the simulation, such as binary rate of links, the size of router queues, or the message rate of sources. After the simulation, ns2 produces an output file with the traces of the simulation (it is noteworthy that these files are generally very large). This file must be the input for *nam*, the visualization tool.

The script files and the output files employed by CNSE users are stored for later retrieval in the **group repository service**. This service will offer a group-view of the files stored (students in a group will see only the files of their group). In order to achieve unlimited storage capacity, and independence of storage location, as well as other features as transparent replication or file cache, this service will be composed over the Virtual File System being developed in Grid4All.

The **simulation service** is in charge of carrying out the simulations of the networking scenarios described in the script files provided by users. The environment can also employ the instances of this service no matter what is the organization that provides them in the grid. This opens up the possibility of carrying out a large number of simulations in a reasonable period of time by using multiple instances of the service provided by other organizations.

The statistical analysis of the data generated by simulations can be carried out by the **analysis service**. Again, the fact that CNSE can access to the instances of this service shared by any organization in the grid assures that the possibilities of performing analysis is not limited by local computational resources. Further, the analysis service is decoupled from the internal implementation of the simulation service, and is independent of the location of the service, since it only uses the trace files generated by it.

The **registry service** is a registry typically employed in service-oriented grids to enable the discovery of the services shared by the organizations participating in a grid. In the case of CNSE, the index service is used to find a shared repository, a simulation service or an analysis service whenever they are needed. When registering these services, the organizations provide not only their location, but also information that is

employed by the environment to choose the service that best suits its needs. It is expected that these services will be provided by Grid4All infrastructure.

The x-y graphs of the measures taken during simulations and the animations of the simulated network scenarios are generated by the **visualization service**. This service also implements the logic that allows sharing the view of a given graph or animation among the members of a group and that enables them to interact with it. Again, this service is decoupled from the simulation service internal implementation and independent of its location.

The **awareness service** keeps the list of users that are online using the application. Besides, it is responsible for notifying the changes that may happen on the list of online users as well as the main actions performed by them.

The use of collaborative services in CNSE such as the shared repository or the visualization service implies that a reference to the shared context of each group of students in each service must be available for the application. In this way, the application can put users in contact with their group mates and with their associated grid resources. Those references, as well as the identifiers of the users that belong to each group, are stored in the **group context service**.

Finally, it must be taken into account that the access to services provided in a grid is typically done in a secure way. This is why CNSE, as many other grid-enabled applications, uses a **credential repository service** to store and retrieve valid user credentials that enable secure access to grid services. Again it is expected that this services will be provided by Grid4All infrastructure.

The services that make up a grid-aware application must be coordinated so that they are accessed according to the functionality that the user demands from the application in every moment. This coordination can be made by light clients employed by users to interact with the application. In the case of CNSE, two such clients are defined: the **participation client** is conceived to allow learners collaborate using the simulator, while the **management client** allows educators to define the groups of students that will use the application. These clients are also graphically represented in Figure 6.

3.2.2 Infrastructure requirements

As mentioned above, the CNSE application will help realize the “collaborative simulation of computer networks” scenario described in Deliverable 4.1, and thus to test the infrastructure requirements reported in section 7.1.2 of that report. The usage of the infrastructure made by the application may be decomposed in two cases: on one hand, the application services make calls on the infrastructure services, or are notified by them; on the other hand, the application services expect to run on an infrastructure that meets certain requirements without making explicit calls to benefit from them (e.g. the provider of a service offers it to a VO but the implementation of the service is independent of this fact).

The CNSE will make explicit usage of the infrastructure in the following situations:

- The **data management** service will be called to store, retrieve and remove files, corresponding to the *ns2* scripts and trace files. These trace files are usually very large and thus arbitrary space should be available on demand. The application services should be unaware of the data management service location, or of performance oriented strategies such as file replication or file cache. This service will be developed in WP3, task 3.1.
- The management client will make use of the **service discovery** infrastructure to locate existing simulation, visualization, awareness or group context services. This infrastructure service will be queried by asking just for a certain functionality (i.e. look for any visualization service) or by adding certain quality of service (QoS) restrictions. In order to do so, providers should be able to publish their service specifying their QoS parameters. This will be specified with the semantic information system proposed in WP2, task 2.3.
- Each of the clients will need to obtain credentials from the **authentication** infrastructure. Each of the services may have to invoke other services (either application or infrastructure services) on behalf of the user, and hence **delegation** mechanisms should be supported. These functions are associated to Virtual Organizations, and hence will be tacked by WP2.

Besides, the CNSE may make implicit usage of the following infrastructure requirements:

- The computational resource upon which a simulation is running fails should be **monitored**. If it fails, it should be restored and the simulation launched again, to achieve **fault tolerance**.
- Similar arguments can be given concerning the storage of files.

In general, both aspects are more related to the service and network components, and hence will mostly concern WP1.

3.2.3 Infrastructure API

According to what is described in the previous section, the application services of CNSE will call the infrastructure as follows.

The **data management** service (WP3, Task 3.1) should support the following methods:

- `fileId = store(fileName, byteStream)`
  - `fileName` is the file name
  - `byteStream` is the binary data in a file
  - `fileId` is an identifier relevant for the storage service
  - *Stores some data in a file in the storage service, if allowed*
- `byteStream = retrieve(fileId)`
  - `fileId` is an identifier relevant for the storage service
  - `byteStream` is the binary data in a file
  - *Retrieves the data of a file in the storage service, if allowed*
- `remove(fileId)`
  - `fileId` is an identifier relevant for the storage service
  - *Removes a file in the storage service, if allowed*
- `fileList = list()`
  - `fileList` is an array of `[fileId,fileName]` pairs
  - *Lists the files in the storage service that correspond to the current VO*

Other methods relating a more elaborated view of the storage system as a Virtual Filesystem (e.g. managing directories, links, and permissions) could also be used, but are not critical.

The **service discovery** service (WP2, task 2.3) should support the following methods:

- `serviceList = find(query)`
  - `query` is a query performed in a query language[1] specifying the functionality of the service, and optionally some QoS restrictions
  - `serviceList` is a list of service identifiers[2]
  - *Finds the services that meet the description of functionality and QoS*

The **authentication** service (management of VOs, WP2) should support the following methods:

- `credentials = getCredentials(user, password, lifetime)`
  - `user` is the user textual identifier
  - `password` is the user password
  - `lifetime` is the duration of the proxy certificate to be used in this session
  - `credentials` are the proxy credentials
  - *Retrieves proxy credentials for a user. These credentials form a chain of trust of signed certificates, along with their associated private keys, where applicable. The first certificate will be issued for this session, with a short duration, and signed by the private key of the user. His public key appears in the second certificate, the long term end-user certificate, which is signed by a Certification Authority. The third certificate (optional) contains the public*

[1] For example, XPath

[2] The specific structure of a service identifier needs to be defined. For example, in WSRF each service is univocally located by an End Point Reference (EPR), which contains the service Universal Resource Identifier (URI, similar to a URL) and the resource key, so that the EPR allows to contact a "stateful service".

key of the Certification Authority, signed by it. This Certification Authority must be widely trusted.

- `storeCredentials(user, password, credentials)`
  - `user` is the user textual identifier
  - `password` is the user password
  - `credentials` are the user credentials
  - *Stores user credentials, possibly generated by other Certification Authority*
- `destroyCredentials(user, password)`
  - `user` is the user textual identifier
  - `password` is the user password
  - *Destroys all user credentials*
- `certificates = getCertificates(user)`
  - `user` is the user textual identifier
  - `certificates` are the user certificates
  - *Retrieves certificates associated to a user (containing the public keys). These credentials form a chain of trust of signed certificates, but without private keys.*

3.3 Collaborative File Sharing

The Collaborative File Sharing (CFS) application is a 'shared workspace' system which supports document sharing, asynchronous discussions, task list management and a private address book. It is related to all scenarios with needs in sharing information among users.

3.3.1 Description of the application

When thinking about the design of a collaborative application, one of the first ideas that come up is to use a single server in a centralized manner. However, this centralized approach does not fit in an autonomic and dynamic environment such as the one Grid4All is intended for. Therefore, the solution that better fits this scenario is a decentralized system.

Decentralized systems are characterized by the non-existence of a server or any privileged node with more information than others. Nodes in a decentralized system are at the same level and act both as client and as server at the same time. The communication is carried out directly from a peer to another peer with not external coordination. Some strong points in this kind of systems are reliability, fault tolerance and availability. On the other hand, some other challenges arise and a solution needs to be given. One of them is concerning to concurrent modifications management issues.

There are two main approaches to face concurrent modifications. The first approach avoids them by setting a lock feature[3]. The other approach lets them to happen and solves them later[4]. The first approach adds some restrictions to the user when accessing data. As we want the application to be as transparent as possible, we decided to use the second approach. This second approach also allows the systems to operate when some node is not available or in disconnected mode. A drawback of having many users modifying the same data is that data conflicts may appear. The application must solve these conflicts, when possible, in a transparent manner to the user. In other cases, only user interaction can solve a conflict. The main goal of these techniques is to maintain a consistent data view throughout the system.

The application here described is being built on top of the Grid4All middleware. Since this is a distributed architecture, our application will be decentralized as well. CFS application will benefit from many mechanisms and services provided by the underlying infrastructure, what will simplify its design. Some of these services are data replication, conflict detection and solving, network maintenance and communication issues.

The application is divided in four modules: file sharing, forum, task list and address book.

[3] Also called Pessimistic
[4] Also called Optimistic

The **File Sharing** module provides a distributed repository where users can upload and download files and organize them in folders. Files are versioned. In fact, every modification in a file creates a new version of the file. Any user can retrieve any file or any version of a file. There are mechanisms for rating a version of a file or for posting a note related to a file. An overall rating is calculated adding up all version ratings. Meta-information is also maintained, as well as, a history of the file that keeps track of the data stored in each version: who and when uploaded it and the relations between versions. History can be displayed as an acyclic graph of versions because there is a parent-child relation between versions.

The concurrent access to files and folders together with its decentralized management may arise conflicts, e.g. uploading a new version from de same file or applying some changes to the directory tree. Grid4All, through Telex, provides a framework for solving some of the conflicts autonomously. The application will use it when possible. Some cases can not be solved transparently and require a human resolution, as a file version conflict. Another functionality of this module is to store some extra information, as who has read a file or post it. All this information provided to users on demand. In addition, a forum can be attached to files. This forum will allow users to discuss aspects related to the file to which it is associated. The functionality of this forum is the same that is explained in the next paragraph.

Second module is a **Forum**. All users may communicate in an asynchronous way using this module. There is a general purpose forum for the application and some others more specific attached to each file, where users can discuss about that specific file. Any user can post. Messages are viewed by all users. Users may either reply a message or open a new thread. In addition, users can check who had read the message and rate them. This extra information allows any user to know what other people in the community think about the message.

The **Task List** module allows a user to put together all tasks that has to do in a short and medium term. An empty space is given to all users so they can put there their tasks. Any other user can view this list. There is a restriction when updating information, only the owner can add and delete items from the list. There is also other information as degree of fulfilment or priority. This module is useful to know what a user is doing and what is planning to do.

Finally, the last module is the **Private Address Book**. Every user has an address book where some private information about other colleagues or friends can be written. The main goal of this module is to provide an always available contact book despite the changes of workstation.

The operations given to users are:

- File Sharing:
  - Related to the directory tree structure
    - Create directory
    - Delete directory
    - Move directory
    - Change directory name
    - List directory content
  - Related to files
    - Create new file, and the initial version will be created
    - Create new version
    - Delete file
    - Move file
    - Change file name
    - Delete versions
    - List versions of a file
    - Read version
  - Related to conflicts
    - Solve conflict
  - Related to Communication
    - Post a new message about a version or file
    - Reply a message about a version or file
    - View a message
    - Rate a version
    - Get information about a file or version
- Forum:
  - Post a new message in the forum
  - Reply a message in the forum

  - View a message
  - Rate a message
  - Modify message
  - Delete message
- Task List
  - Add a new task
  - Delete a task
  - Modify task
  - Change properties of task, as grade of fulfilment
  - View all task from another user
- Private Address Book
  - Add a new contact
  - Delete a contact
  - View contacts

3.3.2 Architecture and application design

The figure below shows the relations between the modules and the underlying framework provided by other WP's. We use Telex and DFS to spread the information and to communicate with other nodes. More information about the use of the framework is detailed below.

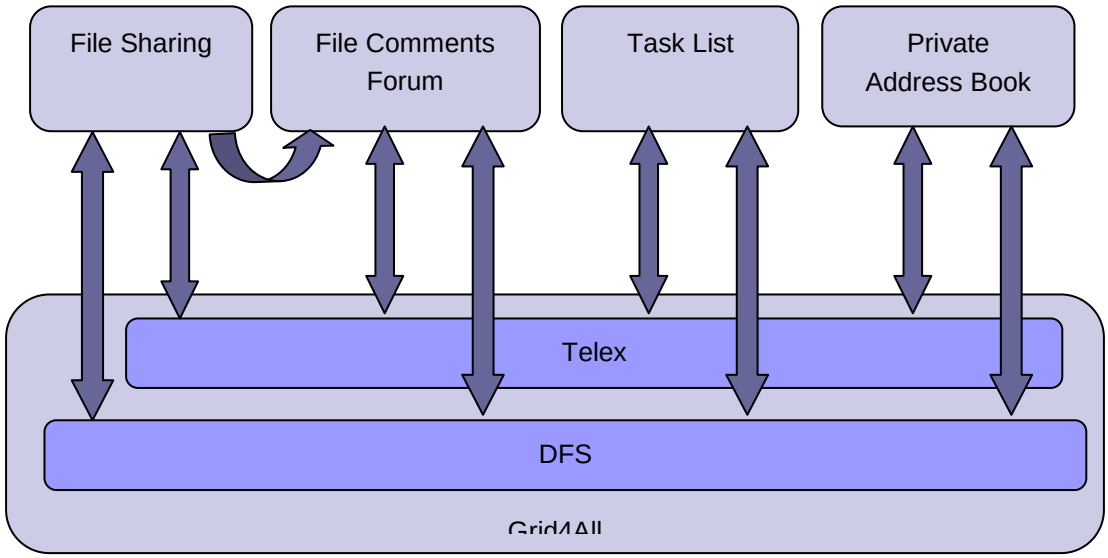

Figure 7: relations between modules and underlying framework

Each of these modules is designed following the model-view-controller (MVC) pattern with some variations since there is some network interaction and an application can react when an event from another node shows up. As each module behaves different from the others, we will go into details separately.

The **File Sharing** module is divided in three layers, as in the MVC pattern. The *view* shows the user the directory tree and other related file information. It also passes requests to the controller and shows the results provided.

The *controller* is responsible for listening requests from the view and from other nodes in the network, processing them and providing a result, which will be displayed in the view. The operations available in this module can be split in three groups. Directory tree change (create directory, move file …), version management (upload new version …) and state review (list directory, view version …). Some of those operations can be conflicting when issued concurrently.

Telex is used for conflict detection and solving. Conflicts produced by operations in directory tree change group are managed by Telex. Operations in version management group cannot be solved automatically since they are related to information with a meaning for human users. So, users should decide which version

is more relevant in case of conflict. Finally, operations in the last group cause no conflict since there are only consultant operations.

DFS is used for storing all data in a distributed way. By using DFS, we can ensure that all information is available to all nodes, so any user from any workstation can retrieve a file version and view it.

The **Forum** module, which is also used in File Sharing module for the file's comments, is also divided in three layers. The view shows a tree of messages from a starting point, usually the root. A user opening the forum sees, firstly, the titles of the messages in a tree structure. When a user clicks on a message, the content is retrieved by the controller from the DFS and appears on the view. Other main functionality in a Forum is adding a message. A user may either reply an existing message or create a new thread. Then, the controller creates a new message with the content specified by the user and stores it on the DFS. The view is updated to show the new message just added. Some other functionalities are provided. Messages can be deleted or modified by an authorized user. In addition, there are information for each message about whom and when it was read, some rating issues and other related information. Conflicts may arise when modifying or deleting a message. We use Telex for detecting them. Modifications can only be carried out by the owner of the message and deleting is carried out by authorized users.

The **Task List** module assigns a space to each user where a user can add some task he has to do, with some properties like grade of fulfilment or priority. Other users can view that list in order to know what the user is working on. Only the owner of a list is able to change the list, either adding a new element, deleting or modifying the state of an existing element. The view catches all requests, either for modifying own or for checking other user's list out. Then it passes the request to the controller and it solves it. Telex manages the data in order to avoid unexpected behaviour when a user does some operation from different workstations. Conflicts may appear when some modifications are done concurrently. In the end, data is stored in the DFS and it is available to all peers. When a user demands a list from another user, it is available on the DFS, so there is no need of further processing or communication.

Finally, the **Private Address Book** module holds a private contact book. Each user has its own book. Only the owner can view its own contact book. He can add and delete contacts anytime. The main point of this module is to have the contact book available over the network, independently from the location of the user. The controller uses Telex in order to avoid changes done concurrently lead to unexpected behaviour. As users can move from workstation to workstation and work off-line, there is a need of controlling conflicts. The data is stored in the DFS and it is kept private. Only the owner can read the stored data.

3.3.3 Infrastructure requirements

The application will use following frameworks:

- **DFS**: The DFS framework should provide some mechanisms for supplying and storing data in a reliable way, disregarding the state of other nodes. There should be some replication techniques in order to have complete availability, reliability, efficiency and full fault tolerance.
- **Telex**: Telex should provide to our application a way for detecting and solving conflicts. Given actions and constraints, it should able to find a sound schedule, i.e. a schedule of operations with no conflicts. In addition, there should be a mechanism for spreading operations throughout nodes. The order the operations are received is not important since the reconciler will update the schedule that should be executed as new operations are received.
- **Authentication**: The application needs a service of authentication that ensures the identity of a user univocally. When working in a collaborative way, it is good to have some mechanisms for attributing the actions to a user and knowing who did each operation.

3.3.4 Infrastructure API

According to the requirements above described, the calls to the infrastructure will be as follow:

- DFS:
  - Create File
  - Remove File
  - Create Directory
  - Remove Directory

  - Read a File
  - List Content of a Directory
- Telex:
  - Add Operation
  - Add constraint
  - Get sound schedule
- Authentication
  - Authenticate
  - Check permissions for a user on something

3.4 Shared Calendar

The Shared Calendar (SC) application aims to help people organizing their agenda in a collaborative way. In fact, SC allows people to create and manage private events as well as group meetings. Thus, it is related to the deliverable 4.15 scenarios where people need to meet or to coordinate some tasks. Such scenarios are described in sections 5.1.4 “Distributed collaborative software project development”, 5.3.2 “Live sessions using collaborative tools”, 5.3.3 “Collaborative software development” and 5.4.1 “Communities”.

The SC application in the Grid4All environment benefits from many mechanisms and services such as data replication, persistency and consistency model. It overcomes the communication, network management and access control issues. Besides, it provides communities awareness and Virtual Organisation (VO) management layer.

3.4.1 Description of the application

Communities and individuals can use the SC application to hold professional and personal activities. One shares his calendar and allows other people to modify his planning by adding new events. To cope with reliability and availability issues the shared agenda and the meeting information are replicated. Moreover a peer-to-peer environment like Grid4All environment brings more scalability, flexibility and autonomy for each node.

Additionally, invitees can modify events they attend concurrently. Therefore SC application has to deal with consistency issues. Basically there are two main ways to handle consistency *pessimistic replication (PR)* approaches and *optimistic replication (OR)* approaches[6].

PR can hardly be achieved in dynamic systems such as Grid4All environment. Most of all accessing data becomes a bottleneck for the system. Besides, PR doesn’t allow disconnected work and isolated work that are useful for such collaboration. Indeed one can create and manage his agenda without needing to be connected; and one can choose not to be overwhelmed by others modifications for a while. In opposition to PR, optimistic replication overcomes the bottleneck in accessing data. In addition OR allows disconnected and isolated work. OR requires nontrivial reconciliation protocols to achieve eventual consistency. Furthermore, it is more complicated to design such algorithm in a dynamic large-scale environment like Grid4All.

The Data Storage layer of Gid4All infrastructure hides the replication and the communication issues from the application level. The Semantic Store (SS) provides mechanisms to achieve consistency in peer-to-peer environment. Additionally, SS provides mechanisms that allow undoing and redoing operations.

3.4.2 Operation principle

Considering the rationale stated above, the shared calendar application will be developed using the SS. Before describing the operation principle we must introduce the outline of the Semantic Store. The SS is based on the Action-Constraint Framework ACF. ACF is a log-based optimistic replication approach, where “actions” are application atomic operations. “Constraints” are relations between actions. The application sets the constraints to express its semantics and to capture user intention.

[5] Deliverable 4.1: Specification of scenarios, user requirements, and infrastructure requirements
[6] *Optimistic Replication.* Yasushi Saito and Marc Shapiro. Computing Surveys, vol. 37, n°1, pp. 42-81, March 2005.

As said before, the application provides users a way to manage their activities. Each user has his own and independent calendar filled with private events. Additionally he can organize meeting with other collaborators. He creates a “meeting object”, shares it, and notifies invitees of their invitation. For that purpose, he creates an operation (action) on their respective calendar. Thus, he has to import the invitees’ calendar.

When one receives an invitation he can accept it or decline it. If he accepts it, he can collaborate to hold the meeting: he can invite other users, and modify the meeting time, and location. For that purpose he creates operations (actions) on the corresponding “meeting object” concurrently with other invitees. Consequently conflicts may appear.

As we are in an optimistic replicated environment, those actions are “tentative” until they are “committed”. In case of conflict some of them are “aborted”.

For more detail on the SC design see the deliverable 3.1 “Requirements analysis, design and implementation plan of Grid4All data storage and sharing facilities: Collaborative applications”.

3.4.3 Functionalities:

The SC provides the following functionalities:

Meeting management:

- Create an event:
  - Create the “object” event.
- Publish an event:
  - Share it with a group without inviting anybody.
- Import a user document:
  - To invite other people to an event
  - Needs the approval of that user.
- Invite a user:
  - Ask a user to attend the event.
  - Share the event “object” with the invited user
  - Notify him of the invitation on his agenda
  - Needs the approval of that user
- Cancel invitation:
  - Ask a user to cancel his coming to a meeting
  - Needs the approval of the cancelled user
- Allocate room
  - Choose a specific room for a meeting
  - Or ask for an available room
- Set/update meeting time
  - Propose a time for a meeting
  - Needs the approval of all invitees.
- Cancel event
  - Propose to cancel an event
  - Needs the approval of all invitees.

- Accept /decline mechanism:
  - To vote on one of the previous actions if required

Collaborator management:

- Get group list and user profiles
- Invite a user to collaborate
  - Ask user to give someone read/write access (share) to his calendar.
- Stop collaboration with a user
  - Remove the read/write access on one’s calendar for a user.
- Accept/decline an invitation for collaboration

- Give or decline read/write access on own calendar.

We can define different policies for meeting modification like priority policies. For instance, one that only allows the creator to modify meeting information, and to add/cancel invitations.

3.4.4 Architecture:

Figure 8 shows the main components of the shared calendar application, and their interaction with the environment.

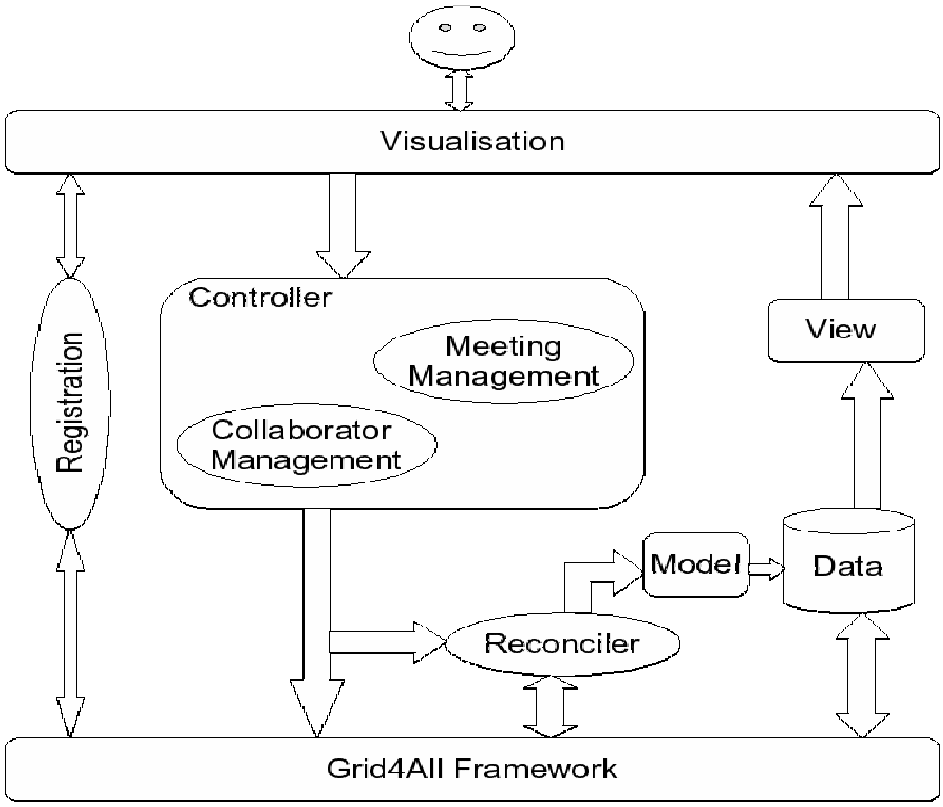

Figure 8. Shared Calendar architecture and interactions

The application is based on the model-view-controller design. A **Visualisation** layer interfaces the user with the SC. A **Registration** component allows user to sign in. This registration may be hidden from the user.

The **Data** component is the materialization of the information the application needs: logs, meetings information, and collaborators' profiles, etc.

A **Controller** component captures user's modifications through the **Visualisation** layer. It generates the corresponding actions using the meeting and collaborators management functionalities.

The **Reconciler** component gets local and remote actions and compute correct schedules.

The **Model** component takes those schedules and executes them on the **Data** component.

Finally the **View** transforms the **Data** state into visual information.

3.4.5 Infrastructure requirements

The application components interact with the Grid4All framework in different ways and at different levels. Those requirements and interactions with WPs are described in the following section.

Registration component requires:

- Authentication services: WP2

    - Get the user identifier
- Login service: WP3, Semantic Store
    - Login session: Inform the SS of the current user of the application.

Data component is replicated and persistent so one can keep his documents even if he works on different computers. It contains among other things users' logs. Thus, **Data** component requires:

- Replication and communication mechanism: WP1, WP3
    - Hidden by WP3
- Log constitution and management: WP3
- Resource management services: WP2
    - May be hidden by WP3

Controller component generates actions recorded and replicated on logs. It requires:

- Log constitution and management: WP3
- Replication and communication mechanism: WP1, WP3
- Resource management services: WP2
    - May be hidden by WP3

Some **Meeting Management** functionalities require more services:

- Create an event:
    - Create a Telex document: WP3
- Publish an event:
    - Access control services: WP3 (an event is a special file)-WP2
- Import a user document:
    - Get document information (Yellow pages): WP2
    - Import the document: WP3
- Accept /decline mechanism:
    - To vote on someone actions: WP3
    - Decision notification:
        - Communications: WP3, WP1

Similarly, **Collaborator Management** needs:

- Get group list and users' profiles:
    - Yellow pages and group awareness: WP2
- Invite a user to collaborate
    - Access control on Telex Document: WP2, WP3
    - Decision notification:
        - Communications: WP3, WP1
- Stop collaboration with a user
    - Access control on Telex Document: WP2, WP3
    - Decision notification:
        - Communications: WP3, WP1
- Accept/decline an invitation for collaboration
    - Access control on Telex Document: WP2, WP3
    - Decision notification: Communications (WP3, WP1)

Finally, **Reconciler** component requires:

- Reconciliation and scheduling services: WP3, Semantic Store
- Collaborators' status per Telex document: WP3.

3.4.6 Infrastructure API

To handle requirements described above, we propose the following infrastructure call

API:

- Authentication:
  - Authenticate
- Login services:
  - Launch (application, user)
- Group awareness:
  - Yellow pages (VO)
- Access control:
  - Set access control (W/R, user, document)
- Resource management:
  - Get (resource)
- Document access:
  - Import a document
  - Export a document
  - Open/Close a document
  - Read/Write in a document
- Telex Document management, reconciliation, scheduling services and collaborators' status:
  - See Telex API in the deliverable 3.1 "Requirements analysis, design and implementation plan of Grid4All data storage and sharing facilities: Semantic Store"
- Notification:
  - Send/receive messages

4. Selective transparency mechanisms in applications

The middleware layer seeks to hide the complexity of the underlying distributed environment and to mask the heterogeneity of computational systems (different operating systems, programming languages, network technologies) involved. However we can find different types of applications requiring some abstract API while others may require a more detailed API in specific parts or even just ignore some other parts. Programmers considering to port or develop applications to the Grid4All environment may need API with diverse levels, may be willing to share code among several applications and in general have a certain choice to be able to reach a trade-off between complexity and control. Consequently, this leads us to consider transparency, a concept playing an important role when designing middleware systems.

There are many definitions of distributed systems. For the purpose of Grid4all, this general definition is sufficient: "*A distributed system is a collection of independent computers that appears to its users as a single coherent system"* [24]. Two important aspects appear in this definition: (1) This system consists of computers (more generally: components) that are autonomous, and (2) users ideally should think that they are using a single system. In fact, this can only be achieved at the expense of pieces of software that hide the details and implications of distribution. In other words, the price for handling or hiding distribution is complexity and lack of control on aspects that may be handled and hidden from applications at lower layers of software.

From the point of view of users, it is desirable that in most of the cases the components collaborate in a distributed environment without having to be aware of all these details, but this is not always possible or desirable.

When a distributed system is able to present itself to users and applications as if it were only a single computer, it is said to be **transparent**.

The ANSA Reference Manual [25] and the International Organization for Standardization Reference Model for Open Distributed Processing (RM-ODP) [26] identified the following forms of distribution transparency:

- Access transparency: hide differences in data representation and how a resource is accessed. For example, components may run different operating systems have their own local file-naming conventions, etc.
- Location transparency: enables resources to be accessed without knowledge of their physical or network location. Location transparency is mainly achieved through naming. An example is the URL http://www.grid4all.eu/, which does not provide any information about the location of the grid4all web server.
- Migration transparency: when a resource can be moved without affecting how the resource is accessed.
- Relocation transparency: hide that a resource is being moved to another location while it is being accessed (without either the user or the application noticing it). An example of this transparency is when a user is able to continue using its wireless laptop while moving from one place to another without ever being disconnected.
- Replication transparency: enables the use of multiple instances of a resource without knowledge of the replicas by users or application programmers. This transparency is used to increase reliability and performance.
- Concurrency transparency: enables several processes to competitively use shared resources without interference between them.
- Failure transparency: hide to users or applications that a resource fails to work properly. It also hides the recovery of the resource from the failure.

Distribution transparency is generally considered preferable for any distributed system. However, there are situations in which to hide all distribution aspects to users is not a good idea. An example is when a user is subscribed to a news service to get the news of the day at 8:00 A.M. local time. If the user is in a different time zone due to a business trip he/she will receive the news at a different local time. Likewise, if the application a user is using involves the access to information in two computers at two ends of the world, the delay due to signal transmission and due to processing of the intermediate switches will affect the performance of the application.

The degree of transparency can also influence the performance of the system when a failure occurs. Many Internet applications try to contact a server several times before to give up in their attempt. Consequently, the mask of transient failures before trying another server may slow down the system.

As it is impossible to achieve full transparency, it may be a good decision to provide different degrees of transparency. Every application will decide the degree of transparency that suits its users. Having to deal with distribution will help users to have a better understanding of the behaviour of a distributed system, resulting in users being better prepared to deal with any unexpected behaviour.

Selective transparency refers to the possibility that users have to either abstract from some aspect of a system provided by a transparent mechanism or interact directly and have more control over that aspect. Therefore, selective transparency offers the freedom to skip the complexity from those aspects where default behaviour is sufficient, concentrating on handling the details of other aspects that can be better solved at a higher level. This makes application development for distributed systems only slightly more complex in cases where optimizations are needed or intended.

4.1 Selective transparency in Grid4All

Grid4All aims to be an infrastructure built from the resources provided by participants (individuals, organizations and utility computing providers). In such a context, different applications will have different dependability requirements, requiring the selection and configuration of different dependability mechanisms.

An example of selective transparency in the context of Grid4All appears when a group of users wants to collaboratively share files. The easiest and most transparent solution is to allow users to use its local file browser to access the shared files. This local file browser is transparently connected to grid4all to allow the sharing of files with other users. This can result in the user perceiving that, from time to time, the application has or produces an awkward behaviour. For example, if two users modify the same file concurrently, there is a conflict. To solve the conflict, a user or a resolver application will select one of the files and discard the other. If the user is not informed that a conflict has occurred and the way it has been resolved it will not understand why the file does not contain the information that was previously introduced.

By relaxing the level of transparency a little, users can continue using its file browser with a monitor tool that informs about conflicting operations and, if support is provided, its resolution. This allows users to better understand the behaviour of the application with only the extra cost of having to develop the monitor.

In WP1 and WP2 the infrastructure is also proposing two related mechanisms that are also addressed at reducing the complexity by hiding some details: abstraction and aggregation. Abstraction is a mechanism to reduce and factor out details to focus on a few relevant concepts at a time. For instance, abstraction of failure models permit to treat multiple types of failures as a simple fail-stop model hiding the specific details of each failure model. Aggregation is a mechanism to combine multiple sources of information. An example is the Market Information Service in WP2 where one agent can consult the average price of a storage resource, which is an aggregated value hiding the potentially large collection of different prices for storage on many peers. This helps to isolate applications from the potentially large scale in number of participants on a peer-to-peer network. These two mechanisms can also be seen as selective transparencies provided by some middleware, hiding specific details to transform information into a simpler or lighter representation.

Summarizing it, Grid4All should support different degrees of transparency, i.e. selective transparency. More precisely, it should support different degrees of transparency or combinations of them at the same time. At one end, there will be simple APIs that will help the portability and adaptability of existing applications as well as the building of applications that do not want to deal with the burden of Grid4All distribution. At the other end, there will be a fully detailed API that will allow applications benefit from all capabilities of Grid4All.

Another example is when there is a need for having several copies of a data (either due to reliability, availability or performance reasons). The transparent solution will be: if one copy is changed, that change should be propagated to all copies before allowing any other operation. Depending on the location and number of copies, this operation may take seconds to complete, something that cannot be hidden from users. In addition, it prevents the working in a disconnected mode.

Wide-area distributed services, dynamic environments, mobile computing, disconnected operation mode among other situations require more flexible solutions. Optimistic replication systems [27] can adapt to those situations by reducing the degree of transparency. The degree of transparency will depend on the techniques used to solve conflicts occurring during concurrent actions. Predefined policies may be the most transparent techniques, like Thomas's write rule [28], where an update can be overriden by a concurrent update by another user. Another group of techniques use semantic resolvers that automatically resolve the conflicts. Finally, the user may be responsible for solving the conflicts. Even using optimistic replication, an application may be designed to use a simplified API that does not pay attention to conflicts or that implement

some generic policy to solve conflicts. In that case, the users will perceive the abnormalities as failures (e.g. temporally unavailable replicas) or as awkward behaviour (e.g. concurrent updates).

Transparency can be applied at different levels. It is very important to identify the right level or layer that has to deal with every transparency. If not, it can result in wrong decision making or inefficiencies [29].

To conclude, applications may assume a certain API, perhaps focusing on controlling the details of some particular aspect (i.e. using a lower level API), while abstracting from other aspects (i.e. using a higher level API). Infrastructure APIs should, at the same time, provide lower level API for applications that want to have detailed control of some aspect, and also provide higher-level API to abstract lower-level details specific to Grid4All. This will facilitate both porting and optimizing applications to the Grid4All environment. Futhermore, this intermediate middleware, providing a higher level API as a result of some specific behaviour to handle some issue at the lower-level API, is also an opportunity to share common code among several applications that handle some issues in the same way.

5. Conclusions

This report has presented the benefits of Service Oriented Computing as a programming model for Grid4All. Some of service properties, such as self description, platform and language independence, low coupling and coarse granularity, make the paradigm suitable for developing grid applications, and several specifications have emerged providing standard ways to develop stateful services and to deploy them in a grid. Further, the relationship between SOC and Fractal has been explored, involving the Service Component Architecture (SCA) framework, which is promoted by several large software and hardware vendors. In this framework, components can be wrapped as services, and they can also invoke web services.

Then, the report discussed four different applications that illustrate the possibilities of Grid4All technology: the eMeeting application allows users to interacting in separate virtual rooms, using text, voice, data and video. The Collaborative Network Simulator Environment supports learning activities that are intensive in network simulation, typical of advanced Computer Network courses, but requiring many computational and storage resources. The Collaborative File Sharing application will allow users to have a unified view of a distributed file system, and to share and version files on it. Finally, the Shared Calendar application will allow participants to share their own agenda, invite others to meetings, or alter other people's agendas.

Interestingly, some of the applications can be used on their own or as part of a broader application, due to the possibility to compose services made of other existing services. For example, the Collaborative File Sharing application proposed in section 3.4 could well replace the group repository service in the Collaborative Network Simulation Environment. Similarly, though not included explicitly in section 3.3, the eMeeting application or the Shared Calendar could also be used in the CNSE in order to have synchronous meetings (in case of pure distance learning), or to set up the work agenda between students.

This report has also presented a discussion on selective transparency. Since Grid4All is intended for an autonomic and dynamic environment, infrastructure developers should support different degrees of transparency. Furthermore, application developers also have to deal with different degrees of transparency and decide which are the aspects the application must deal with and which aspects are left to be handled by the underlying infrastructure.

6. References

[1] BEA Systems, Cape Clear Software, IBM, Interface21, IONA Technologies PLC, Oracle, Primeton Technologies Ltd, Progress Software, Red Hat Inc., Rogue Wave Software, SAP AG, Siebel Systems, Software AG, Sun Microsystems, Sybase, and TIBCO Software Inc., Nov. 2006, http://www-128.ibm.com/developerworks/library/specification/ws-sca.

[2] BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, and Siebel Systems, "Bussiness Process Execution Language for Web Services version 1.1", May 2003, http://www.siebel.com/bpel.

[3] Chung, J.-Y., Lin, K.-J., and Mathieu, R. G., "Web services computing: advancing software interoperability", *Computer*, vol. 36, no. 10, Oct. 2003, pp. 35-37.

[4] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S., "Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI", *IEEE Internet Computing*, vol. 6, no. 2, June 2002,

[5] Dufrčne, D. and Seinturier, L., "Bridging component frameworks: Fractal & SCA", Presentation at ObjectWeb, June 2006, http://www.objectweb.org/phorum/download.php/16,309/BridgingComponentFrameworks_G_Dufrene.pdf.

[6] Dünnweber, J., Gorlatch, S., Baude, F., Legrand, V., and Parlavantzas, N., "Towards Automatic Creation ofWeb Services for Grid Component Composition", *Proceedings of the CoreGRID WP7 Workshop on Grid Systems, Tools and Environments,* Oct. 2005.

[7] Foster, I. and Kesselman, C. Concepts and Architecture. In: *The Grid 2: blueprint for a future computing infrastructure*, eds. Foster, I. and Kesselman, C. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2004, pp. 37-64.

[8] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S., "Grid services for distributed system integration", *Computer*, vol. 35, no. 6, June 2002, pp. 37-46.

[9] Heineman, G. T. and Council, W. T. *Component-based software engineering: putting the pieces together*, Boston, MA, USA: Addison-Wesley, 2001.

[10] IBM Corporation, "Web Services Flow Language (WSFL 1.0)", May 2001, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[11] Krishnan, S. and Gannon, D., "XCAT3: a framework for CCA components as OGSA services", *Proceedings of the 9th High-Level Parallel Programming Models and Supportive Environments (HIPS 2004), held in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS 2004),* Santa Fe, NM, USA, Apr. 2004, pp. 90-97.

[12] The OGF home page, http://www.ogf.org/, last visited: Mar. 2007.

[13] Open Grid Forum (OGF), "Open Grid Services Architecture (OGSA) version 1.5", July 2006, http://www.ogf.org/documents/GFD.80.pdf.

[14] Organization for the Advancement of Structured Information (OASIS), "Web Services for Remote Portlets specification, version 1.0", Aug. 2003, http://www.oasis-open.org/committees/wsrp.

[15] Organization for the Advancement of Structured Information (OASIS), "UDDI version 3.0.2", Oct. 2004, http://uddi.org/pubs/uddi_v3.htm.

[16] OASIS Web Service Resource Framework v1.2 standard, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf, last visited: Jan. 2007.

[17] Papazoglou, M. P. and Georgakopoulos, D., "Service-oriented computing", *Communications of the ACM*, vol. 46, no. 10, Oct. 2003, pp. 25-28.

[18] Szyperski, C., "Component technology - what, where and how?", *Proceedings of the 25th International Conference on Software Engineering (ICSE'03),* May 2003, pp. 684-693.

[19] Vinoski, S., "Web services interaction models, part 1: current practice", *IEEE Internet Computing*, vol. 6, no. 3, June 2002, pp. 89-91.

[20] Vinoski, S., "Putting the "web" into web services: web services interaction models, part 2", *IEEE Internet Computing*, vol. 6, no. 4, July 2002, pp. 90-92.

[21] World Wide Web Consortium (W3C), "Web Services Description Language (WSDL) 1.1", Mar. 2001, http://www.w3.org/TR/wsdl.html.

[22] World Wide Web Consortium (W3C), "SOAP version 1.2, part 1: messaging framework", June 2003, http://www.w3.org/TR/soap12-part1/.

[23] Zhang, L. J., Li, H., and Lam, H., "Services computing: grid applications for today", *IT Professional*, vol. 6, no. 4, July 2004, pp. 3-7.

[24] Tanenbaum, A., Van Steen, (2997). M. Distributed Systems: Principles and Paradigms, 2/E. Prentice Hall. ISBN-13: 9780132392273.

[25] ANSA (1989). The advanced Network Systems Architecture (ANSA) Reference Manual. Castle Hill, Cambridge

England: Architecture Project Management.

[26] International Organization for Standardization (1995). Open Distributed Processing Reference Model. ISO/IEC IS 10746, International Organization for Standardization.

[27] Y. Saito and M. Shapiro (2005). Optimistic Replication, ACM Computing Surveys, vol. 37, no. 1, Mar. 2005, pp. 42–81.

[28] Birell, A. D., Levin, R., Needham, R. M., Schroeder, M. D. (1982). Grapevine: An exercise in distributed computing. Comm. ACM25, 4 (Feb.), 260–274.

[29] Saltzer, J. H., Reed, D. P, Clark, D. C. (1984). End-To-End Arguments In System Design. ACM Transactions on Computer Systems. Vol 2, Issue 4. pp. 277–288.