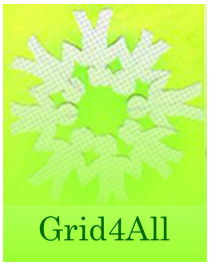


Project no. 034567

Grid4All

Specific Targeted Research Project (STREP)

Thematic Priority 2: Information Society Technologies

D2.4 Prototype of Grid4All resource management services

Due date of deliverable: 01-06-2009

Actual submission date: 25-07-2009

Start date of project: 1 June 2006 Duration: 37 months

Contributors : INRIA, FT, UPC, UPRC

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)	✓

Table of Contents

Table of Contents **1**

Abbreviations used in this document **3**

Grid4All list of participants **4**

1 Executive Summary **5**

2 Introduction **6**

3 Resource Management for democratic grids **7**

3.1 Introduction 7
3.2 Current trends and cloud computing 7
3.3 Integrated usage of Grid4All resource management services 7

4 Management tools for virtual organisations **9**

4.1 Overview and principles 9
4.2 Purpose and features offered 9
4.3 Architecture 10
4.4 Implementation 11
4.5 Configuration 12
4.6 Used technologies 12
4.7 Other related G4A components 12
4.8 Current status 12
4.9 State of art comparable products and specifications 13
4.10 Software 14
4.11 Innovative usages 15
4.12 Conclusions and lessons learnt 15

5 Information Service **16**

5.1 Overview and principles 16
5.2 Purpose and features offered 16
5.2.1 Market advertisement *17*
5.2.2 Application Service advertisement *17*
5.2.3 Querying *17*
5.3 G4A-SIS Architecture 17
5.4 Implementation 18
5.4.1 Market matchmaking *18*
5.4.2 Service matchmaking *19*
5.4.3 Service Selection *19*
5.4.4 The G4A-SIS Registry *20*
5.4.5 The Resource Registry *20*
5.4.6 Service Registry *21*
5.4.7 Querying *22*
5.5 Configuration 22
5.6 Used technologies 22
5.7 Other related G4A components 23
5.8 Current status 23
5.9 State of art comparable products 23
5.9.1 Related work on discovering and selecting traded resources and services in Grid e-markets *24*
5.9.2 Discussions on State of Art and lessons learnt *25*
5.10 Software 25

5.11 Design of a Distributed Information Service 25
5.11.1 State of the Art 26
5.11.2 Principles and requirements 28
5.11.3 Design of the Distributed Semantic Information Service 28
5.11.4 Data Layer 35
5.11.5 Design of the Data Layer 36
5.11.6 Distributed Registration and Retrieval of Application Services 36
5.12 Innovative usages 37
5.12.1 Formal conceptualization and Ontology 37
5.12.2 Automating of semantic annotation process 37
5.12.3 Ranking of matched resources and services 37
5.13 Conclusions and lessons learnt 37

6 Market-based resource allocation 39

6.1 Overview and principles 39
6.2 Purpose and features offered 39
6.3 Architecture 41
6.3.1 Configurable auction server (CAS) and mechanisms 41
6.3.2 Market Information Service (MIS) 43
6.4 Implementation 44
6.4.1 Technical considerations 45
6.5 Configuration 51
6.6 Used technologies 52
6.7 Other related G4A components 52
6.8 Current status 52
6.8.1 CAS and auction mechanisms 53
6.8.2 MIS 53
6.9 State of art comparable results 54
6.9.1 Discussions on state of art 55
6.10 Software 55
6.11 Innovative usages 55
6.12 Conclusions and lessons learnt 57

7 Overall result summary 59

7.1 Software prototypes and status 59
7.1.1 Semantic Information Service 59
7.1.2 Selection Service 59
7.1.3 Market information service 60
7.1.4 Configurable auction server and mechanisms 60

8 Conclusions 62

9 References 63

Abbreviations used in this document

Abbreviation / acronym	Description
G4A	Grid4All
RMS	Resource Management System
SIS	Semantic Information System
VO	Virtual Organization
SLA	Service Level Agreement
LDAP	Lightweight Directory Access Protocol
DCMS	Distributed Component Management System
OGF	Open Grid Forum
ADL	Architecture Description Language
OWL	Ontology Web Language
DHT	Distributed Hash Tables
RDF	Resource Description Framework

Grid4All list of participants

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

1 Executive Summary

This document presents a conclusive report of different components of the management subsystem developed within the Grid4All project. This report extends and completes the previous deliverables D2.2 and D2.3 by describing the final status of design and software prototypes. We distinguish (a) resource management within a virtual organisation (VO) from (b) resource management between (or for) virtual organisations. The first (a) provides tools and services to manage resources and applications under the control of a virtual organisation whereas the second (b) provides support to locate resources and services and to allocate resources and services for purposes of using them within a VO.

As described in D2.2, the project focused on deployment of distributed applications even though a set of other key management tools and services (security infrastructure, membership, discovery and reservation management) have been implemented. This was done to demonstrate that such core management services could be rendered conformant to the Grid4All component-based architecture. ADL-based application deployment tool is completely implemented. To remove restrictions of ADL-based deployment, the project has designed a framework and implemented tools for deployment and dynamic reconfiguration.

The role of the G4A-SIS (Semantic Information System) is to provide services that allow participants in democratic Grids to locate resources and services. Resource lookup is indirect; G4A-SIS is used to locate markets at which resources of desired characteristics are traded. The G4A-SIS has been described in detail in D2.3. This document recapitulates the main features of the G4A-SIS and presents a decentralized design.

We have designed and developed tools that may be used to build market places to trade computational resources. The goals of the project were to identify technology and services that are essential to building open, scalable and extensible resource market places. This has been described in D2.1 and D2.3. A subset of identified tools has been implemented and evaluated; this is described in this report.

The Grid4All DoW has listed two deliverable reports; D2.4 focussing on Grid4All VO management and D2.5 focussing on prototype of Grid4All resource management system. While software deliverables are independently delivered we have preferred to collapse these two reports into one. The reason follows the explanation in the first paragraph. While VO management tools assist VO members and applications executing on VO resources, the Grid4All resource management tools allow (i) VO members to locate and use services executing outside of a (given) VO's management boundary (ii) locate and allocate (in fact lease) resources that are not yet part of a (given) VO. Through this consolidated document we show how these two sets of services interrelate.

2 Introduction

This report describes the implementation of the software components that were designed and implemented to support resource and application management for democratic grids. This document provides an overview of the architecture, main design principles, foreseen usages and the status of the components of the Grid4All management system. As had been explained in D2.2 and D2.3, Grid4All maintains the notion of a Virtual Organisation (VO) that encapsulates resources, members and activities. Resources and members in fact form a VO to implement their joint and coordinated activities.

The document is structured as follows. Chapter 3 provides an overview of resource management and positions it in the context of cloud computing. Chapter 4 describes the main contributions made by the project towards VO management that includes membership, security, and deployment of VO infrastructure, applications and services. Grid4All has focused on deployment, the process by which application life-cycle is maintained. It involves code transfer towards target nodes, installation of components and configuration, connecting and starting components, and continuous maintenance of the application's architecture. In a democratic grid where nodes may leave, arrive and fail, deployment is a continuous task. Grid4All has also designed and implemented a policy-based security infrastructure. This is described in detail within D2.2.

Chapter 5 describes the Semantic Information System (SIS). This system provides services to manage services and markets. The services delivered by the G4A-SIS have been described in detail within D2.3. Chapter 5 summarizes previous work and describes in detail a decentralized design for the SIS; decentralization is imperative for democratic grids where the actors (consumers, providers, resources and services) are characterized by their scale and heterogeneity.

Chapter 6 describes tools for an open resource market place. It describes the architecture and the principle tools that have been designed and implemented. Three aspects need to be pointed out. (i) The project has focussed on a set of technical enablers, but clearly a full-fledged market place requires additional sub-systems similar to any other e-commerce platform. (ii) We have focused on raw ICT resources, but the architecture and tools may be extended for service market places. (ii) The market place serves to allocate resources, i.e. match providers and consumers. It is agnostic to methods and protocols by which the computational resources will be accessed and used. Within Grid4All we rely on the VO management mechanisms provided conjointly by WP1 and WP2 to enable resource consumers to access and remotely use the leased resources.

3 Resource Management for democratic grids

3.1 Introduction

Resource Management System (RMS) is a set of functionalities that distributed computing infrastructures should provide; the goal is for users to locate suitable resources and use the resources, e.g. deploy and execute applications. Grid resources (nodes) are contributed by independent administrative domains with local policies. Node-wide RMSs are subject to local control; security, authorization, usage accounting, etc.

A democratic grid is a socio-technical concept where the user community owns the IT infrastructure; spare resources of heterogeneous networked personal computers and devices are pooled to provide a virtual data centre to form collectively community grids. Democratic grids should allow such diverse resources to inter-operate and should be simple to use, extensible and scalable.

3.2 Current trends and cloud computing

Clouds provide hardware, system software and applications as a service. They give the illusion of infinite computing resources available on demand and where users pay for resources on a usage basis. Public clouds support multi-tenancy through virtualized infrastructures and providers realise economies of scale by spreading costs across clients. Infrastructure clouds provide minimal APIs for clients to allocate virtual machines (VM) with desired system images and configure network addresses. Then,

- Clients deploy and manage applications on these allocated virtual machines using their own methodologies.
- VMs may be dynamically provisioned to deployed applications, either manually or programmatically.

Platform clouds provide managed hosting services with tools to manage application deployment and run-time management. Resource usage is metered and management monitors and enforces SLAs. These platforms use optimisation engines to maximize resource usage. Platform cloud products differ in the infrastructure components such as firewalls, gateways, load balancers, web servers, application servers, file and database servers they provide.

Resource management tools developed within Grid4All may be used even in the context of clouds; autonomic application management, discovery and information service and market-place tools. Grid4All can also be simply extended to absorb key cloud technologies such as virtualization, firewall management.

3.3 Integrated usage of Grid4All resource management services

This section describes a scenario where the different Grid4All resource management components and services are put to use. Assume a school VO has been created by three higher secondary schools to collaborate on a science project to learn volcano physics. Students of the different schools, aided by tutors collaborate to achieve the goals of the project. Each school has reserved a small number of computers for the project. Grid4All VO management tools are used to set-up the VO, allow students to join in as members. They share documents using VOFS, the collaboration tool described in D3.3 and D3.6. The scenario in Figure 1 below is enacted during the project and is briefly described here.

Students are organised in groups and each group executes a volcano simulator application. This application integrates legacy simulation software and redesigned to use Niche, the Distributed Component Management System described in D1.5. Management components monitor simulator tasks and measure progress over time. The complete application is described using the Architecture Description Language (ADL). The application is initially deployed on a subset of the compute nodes currently present (logged in) the VO. Every logged node executes the Grid4All container and the total set of nodes is organized as a peer-to-peer network. The managed volcano simulator requests allocation of new processing capacity using the API of the Reservation Manager, since it has decided (based on monitoring) that the simulation is not making satisfactory progress. The Reservation Manager using the Negotiator component provisions new resources.

The Negotiator identifies candidate auction markets using the G4A-SIS (Semantic Information System) and uses the MIS (Market Information Service) to estimate average and maximum CPU prices. The Negotiator obtains the end point reference to one of the selected auction markets, registers and submits its bid. It waits for the agreement notification. Matched providers are also notified along with information (URLs) that permit the provider to configure its compute nodes such that they join the buying VO (typically by issuing invocations to a bootstrap node on the buying VO).

Once the new resources have joined, the managed volcano simulator deploys additional simulator tasks on the new resources.

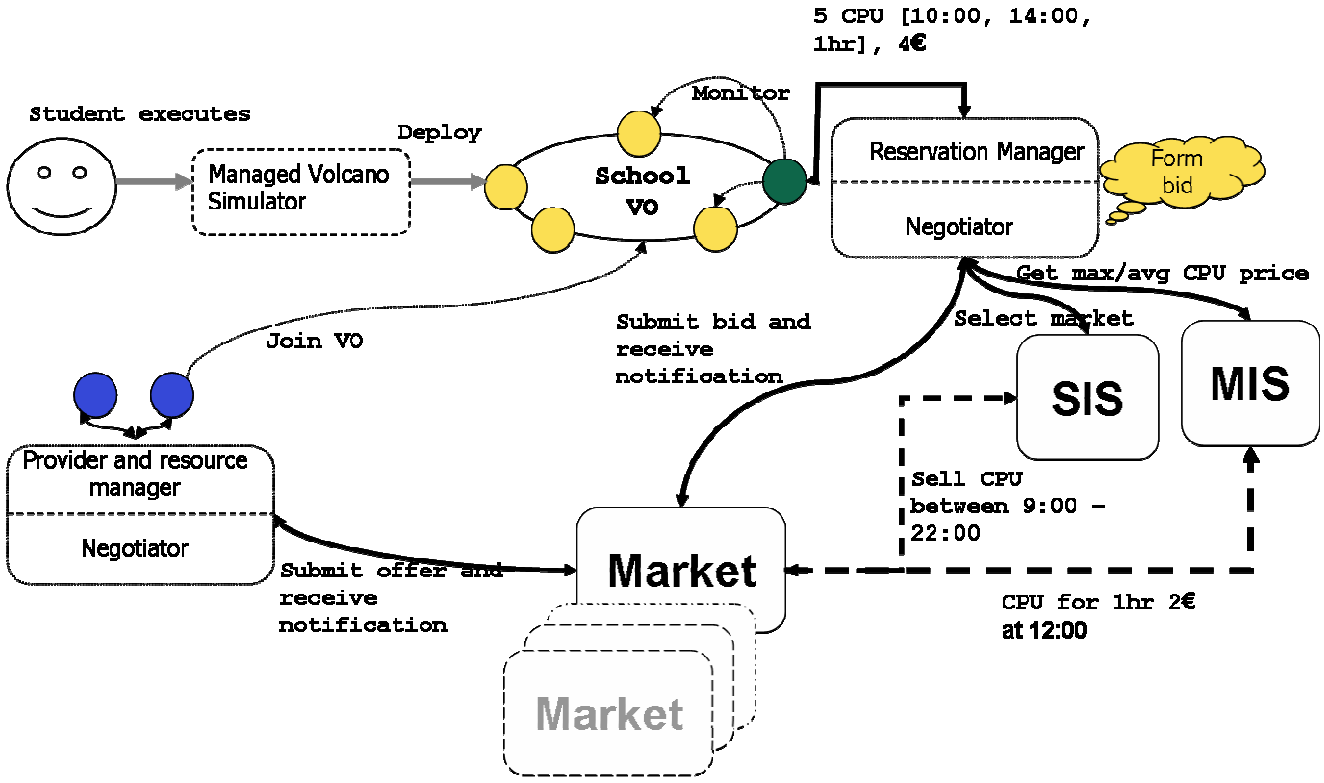


Figure 1 Integrated usage scenario

Chapter 4 describes the VO and application management tools used to set-up the VO, to manage membership, to deploy application and to provision new compute nodes. Chapter 5 describes how the G4A-SIS is used to locate markets where matching compute nodes may be leased. Chapter 6 describes the tools of the market-place that are used to lease compute nodes.

4 Management tools for virtual organisations

4.1 Overview and principles

In Grid4All, VOs are virtual entities formed of users, resources and applications. Users are members who pool and use hardware resources managed by a set of software resources that permit them to execute applications. Management includes creating and deleting VOs, maintaining members and their attributes, discovering, allocating and monitoring VO resources, deploying application code, and enforcing security. The Grid4All VO management tools comprise a set of infrastructure components, libraries and services, which realise the VO abstraction and allow manipulating VO resources and applications.

As described in D2.2, Grid4All follows the architecture-based management approach, which uses architectural models as the basis of deployment, monitoring and reconfiguration activities. This approach provides a high abstraction level for management, hiding the idiosyncrasies of the underlying infrastructure. It enables seamless integration between configuration management and deployment activities, limiting architectural erosion and enabling a more rapid development/deployment cycle.

4.2 Purpose and features offered

Tools are required during the entire life-cycle of a virtual organization: creation, operation and termination. During its operational period, management should enable the following functionality:

Creation of VOs and membership management:

Users should be able to register with a VO, log in/out, and retrieve membership information including on-line status, member roles and policies associated to roles. Users logging in to a VO should be authenticated before admission. In our current design, membership is supported through a programmatic and web interface and relies on a centralized LDAP server to store member information.

User authorization and access control:

VOs should be able to enforce access control policies on their resources, e.g. compute nodes, storage, files and services. Grid4All policy-based security is enabled through a generic API and user-friendly tools to set and update policies. In our implementation, policies are written and stored as XACML (eXtensible Access Control Markup Language) documents.

Discovery and monitoring of resources:

Authorized users (and applications) should be able to discover computational and storage resources present within the VO and allocate them. This resource set is dynamic since physical (or VMs) may join and leave the VO. Our allocation API is implemented using a first-come-first-served resource allocation mechanism. Sophisticated RMS, e.g. policy-based systems, may be implemented using this low-level mechanism.

Resource reservation:

In the normal course of action, users and applications will use computational resources currently available within the VO. When resources are insufficient (immediately or for future times), the reservation service provides a small and easy-to-use API to lease resources for determined periods of time from resource market places. This API hides the complexity of interaction with the market place.

Initial deployment of applications:

Deploying distributed applications is complex and requires knowledge of both application architecture and the infrastructure capabilities and should therefore be supported by tools. A useful starting principle is to separate logical application architecture from its mapping on the infrastructure, in order to enable deploying the application on different infrastructures (e.g., various forms of grids and clouds). In Grid4All, application architectures are described in Fractal ADL (Architecture Description Language) and define: (1) the

application structure in terms of components, their relationships, their bindings and their attributes; (2) packaging information (e.g., names of packages); and (3) resource requirements (CPU, memory, disk space) of individual components. ADL descriptions are submitted to the deployment service that performs the deployment on the physical infrastructure, removing need for human intervention. This service automatically discovers and allocates matching compute nodes, installs dependent software packages, configures and starts the distributed application. Component packaging conforms to OSGi specification.

Dynamic reconfiguration of applications:

ADL-based approaches do not handle application reconfiguration, essential in dynamic environments where applications scale up and down, adapt to resource failures or dynamically locate and link to new component versions; management components are expected to use low-level APIs offered by Niche (D1.5) to reconfigure applications. Whereas the current Niche prototype supports ADL-based initial deployment and a low-level API for programmatic reconfiguration; the *DepOZ* programming framework provides flexible, high-level support for deployment and dynamic reconfiguration. DepOZ can be used to facilitate programming of deployment and dynamic (run-time) reconfiguration in Niche. It provides building blocks to navigate and dynamically modify component structures. Whereas the ADL-based service sequentially deploys components described in the ADL, DepOz supports changeable deployment workflows.

DepOz is made of three parts:

- *FructOz*: a lightweight implementation of the Fractal component model in Oz. FructOz extends Fractal with the ability to model component packages and their dependencies.
- *LactOz*: a library that allows navigating and querying a component structure, much like XPath allows to navigate and query XML documents. However, unlike XPath expressions, LactOz expressions capture the dynamicity of component structures and their values are automatically updated following architecture evolutions.
- *WorkflOz*: a library for defining workflows in a concise and compositional way, taking advantage of Oz's support for higher-order programming. The library provides a set of abstractions that capture the well-known workflow patterns and can be easily extended to capture new patterns.

4.3 Architecture

The VO management system forms a core element of the Grid4All middleware and is decomposed into core services (membership, basic resource management, registry and deployment), the security infrastructure, and two utility services (reservation management, and remote node addition). This is shown in Figure 2.

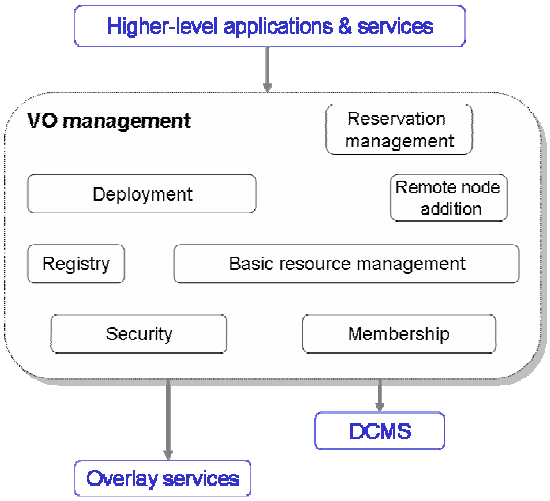


Figure 2 VO management

Membership manages members and their associated information, including roles, their rights and obligations, and their on-line status. *Basic resource management* handles providing resources to the VO, resource discovery, and allocation. *Deployment* relies on declarative application descriptions written in ADL and supports installing, configuring, and activating component-based applications. The *registry* supports registering and looking-up VO services. The *security infrastructure* supports implementing VO-wide authorisation policies. Finally, *reservation management* supports reserving resources in advance, and *remote node addition* supports requesting remote machines to join the VO. These services are completely integrated within the Grid4All architecture.

As explained previously, the *DepOz* is a framework that removes some of the limitations of ADL-based deployment. *DepOz* is organised in two layers as shown in Figure 3. The bottom layer contains *FructOz*, that implements both the Fractal component model and the reference deployment model described in D2.2. This model introduces explicit support for software component packages, their dependencies and local package installation. The top layer provides building blocks for managing FructOz-based distributed systems. It currently contains *LactOz* and *WorkflOz*.

LactOz is used to monitor and navigate dynamic Fractal architectures; *WorkflOz* is used to define and execute deployment and reconfiguration workflows. *DepOz* is an Oz-based development framework built on the Mozart-Oz system and is not implemented (integrated) within the Grid4All architecture; i.e. independent of the Java-based Grid4All middleware stack. Minimal interoperation may be achieved by using the FructOz/Julia bridge that could be used to manipulate Java-based components within FructOz code.

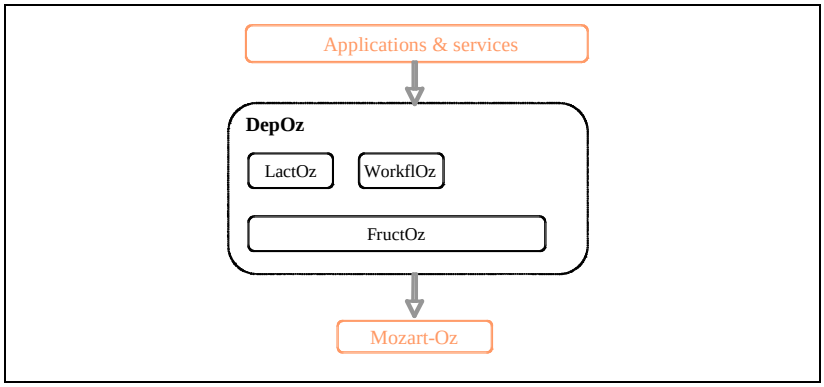


Figure 3 DepOZ dependencies

4.4 Implementation

The VO management tools (described previously) are implemented in Java as a set of Fractal components integrated in the Grid4All container. This container includes Niche[1], the Distributed Component Management System described in D1.5, D6.7 and D4.8. Physically a VO can be considered as a set of Grid4All containers organised as a structured overlay network. Components of Niche runtime environment maintain the overlay. Two container configurations are implemented: the JadeBoot container includes all centralised VO functionality and is used to bootstrap the system and the JadeNode container. A VO is an overlay of one JadeBoot and multiple JadeNode containers[2] that provide run-time support for application components. The Grid4All policy-base security is implemented as a set of components in Java using the Sun's XACML implementation for policies (XACML policy engine).

[1] Niche is a Distributed Component Management System developed in Grid4All to develop, deploy and manage self-managing component-based applications. Niche provides a programming model and corresponding API.
[2] Containers may be deployed on virtual machines as well.

FructOz and LactOz are implemented as libraries written in Oz, and uses deployment and distribution services of the Mozart virtual machine, to package, load, and invoke remote Oz components. WorkflOz is a language designed to specify workflows. It takes the form of a set of high-order functions (i.e. functions that take other functions as parameters) that can be composed to define arbitrary workflows. Workflows are implemented as configurations of FructOz components representing individual tasks.

4.5 Configuration

The VO management system is part of the Java-based Grid4All middleware, and its installation and configuration are described in documentation accompanying the software distribution. *DepOz* relies on installing the Mozart programming system on all nodes and then using the provided libraries to write deployment and reconfiguration tasks. Further details and examples are provided in the associated documentation.

4.6 Used technologies

The management tools use the Fractal component model that provides strong support for building dynamically reconfigurable systems. DepOz uses the Mozart[3] system in order to take advantage of its multi-paradigm programming features. In particular, FructOz relies on the transparent distribution feature to allow Fractal components to be distributed over multiple machines. LactOz and WorkflOz rely on the functional and concurrency features to allow expressing complex tasks in a concise and modular way.

4.7 Other related G4A components

Some of the VO management services and tools use the overlay services provided by Niche, in particular to query and discover nodes within the overlay forming the VO. The high level ADL-based deployment tool described previously uses Niche APIs in the process of component creation, configuration and binding.

The *DepOz* is an evolution over static ADL-based deployment and removes a number of related restrictions. The prototype implementation relies on the general-purpose distribution support provided by the Mozart-Oz platform. It takes no advantage of the overlay-based Niche system, which is specifically designed for volatile and unreliable environments. A natural direction for further work is therefore to extend DepOz to use Niche, combining the robustness and scalability features of the latter with the usability benefits of the former. However, this work requires substantial engineering effort and is beyond the scope of the project.

4.8 Current status

The ADL-based deployment service is implemented and is also employed in the Niche implementation. Basic quantitative evaluations have been reported in D5.2. A proof-of concept prototype of the membership service is implemented as a centralized service. A distributed policy-based security infrastructure is implemented using the Sun's XACML implementation. Proof-of-concept prototype of the Reservation Management API is implemented and uses the APIs of the market place to allocate resources and process successful agreements.

The core features of DepOz are implemented and evaluated on a single machine. DepOz can be extended with libraries and services targeting specialised application classes or execution environments. For instance:

- DepOz can be extended with resource discovery, monitoring and reservation services such as those described previously; these services could also be extended to manage resources provided by clouds.

[3] http://www.mozart-oz.org

- It can be extended with a set of high-order functions that capture deployment and reconfiguration management of recurring application structures, such as master-worker or pipelines. Building self-managed applications would then take the form of composing such functions.

4.9 State of art comparable products and specifications

In VO management, we focus on deployment and management of distributed applications. Nevertheless, we have designed and implemented other VO management functionalities of democratic grids as proof-of-concept prototypes compliant to the common component-based Grid4All architecture, thus increasing the value of the Grid4All platform. We have reused where possible existing solutions to adapt them to the common component-based Grid4All architecture. The innovation mainly lies in the componentised structure; this enhances configurability and extensibility. As an example, the centralised membership service could be replaced by a decentralised version by simply changing ADL descriptions.

Our essential contribution is in hosted application management and in particular deployment and dynamic reconfiguration. This is a problem that is being addressed by a growing body of work in academia and industry. It is recently claimed that to make Cloud computing a reality, the following are required (a) modular, dynamically-scalable, on-demand application infrastructure components (b) development languages and management systems (c) separation of application's logical design from its physical deployment characteristics.

Currently available support for deployment and dynamic reconfiguration may be evaluated based on the following requirements:

- **Usability:**

  Tools must be easy to use, either for initial deployment or to specify adaptive behaviours to reconfigure applications. For example, declarative deployment approaches are more usable than procedural approaches.
- **Flexibility:**

  The platform should support a large range of deployment/reconfiguration activities. In particular, it should support both static and dynamic reconfigurations and it should allow fine-grained control of when the activities are triggered (e.g., to support delaying the deployment of individual components up to the point where they are needed).
- **Generality:**

  The provided support should be applicable to different applications in various domains. For example, today's Cloud platforms indeed offer support for managing hosted applications, but they are specialized; e.g. for 3-tier web applications.

Existing work may be categorized as:

(a) Static software architecture description such as [Deng05, Flissi06, Kon05, Lan05, Mikic02, Wang06]

(b) Workflow languages to describe deployment processes, such as SmartFrog [Goldsack03], Workflakes [Valetto03], Andrea [Rowanhill07], Plush [Albrecht07], and the use of the BPWS4J workflow engine with the IBM Tivoli deployment engine [Badonnel04].

(c) Application descriptions that include constraints and use constraint solving algorithms to dynamically determine deployment targets, such as [Bastarrica98, Malek05, Tibermacine07, Mikic05]

(d) AI methods to generate deployment processes, such as [Arshad07, Kichkaylo04, Heydarnoori06]

We have compared these systems and DepOz against the previously listed requirements. Table 1 provides a simplified view of this evaluation. Systems of category (a), including our ADL-based deployment service are easy to use but allow limited reuse since the built-in workflow may not be dynamically modified. Systems of category (b) support workflows, but unlike DepOz, workflows cannot be composed or parameterised. This limits reusability. Systems of category (c) provide limited generality as they focus on specific deployment problems and objectives (e.g. such as ensuring a certain degree of availability). They must be

complemented with additional mechanisms to deal correctly with exceptional conditions or to ensure synchronization conditions, not enforceable within their scheduling framework. Systems in category (d) expose a high abstraction level for reconfiguration. But the associated high computational cost still remains a barrier other than for simple system configurations. While integrating Artificial Intelligence planning techniques in workflow management systems [Moreno02] is a promising arena, expressivity and scalability are still open issues.

DepOz combines a high degree of flexibility and generality through its use of the Fractal component model with a high abstraction level through its use of higher-order, parameterised workflows that can be composed and reused in different contexts. DepOz thus satisfies the selected requirements in a balanced way and can serve as the base for integrating techniques from other approaches, such as constraint-based techniques.

	Flexibility	Usability	Generality
Static architecture-based	▽	▲	▲
workflow-based	▲	▽	▲
constraint-based	●	▲	▽
AI planner-based	●	▲	▽
DepOz	▲	●	▲

▲ *high*
● *sufficient*
▽ *low*

Table 1 Comparison of deployment technologies

Deployment is the focus of several standardisation efforts, including the OASIS Solution Deployment Descriptor Specification (SSD), GGF's Configuration, Description, Deployment and Lifecycle Management Specification (CDDLM), and DMTF's Open Virtualization Format (OVF). SDD standardises information about software packages and their requirements, and addresses heterogeneity of hosting platforms. Fractal-based architecture descriptions play a similar role in our systems. CDDLM proposed a deployment solution based on SmartFrog but adapted to Web service standards; the working group has now closed. OVF targets the packaging and distribution of virtual machines; this capability is not currently supported by our implementation, but, our architecture allows such an extension, and OVF is relevant. Other related standards are the OASIS Web Services Distributed Management (WSDM), Web Services Business Process Execution Language (WS-BPEL), and GGF's Job Submission Description Language (JSDL). WSDM standardises management and monitoring of services. The Fractal management interface is richer since it enables changing the configuration of composite components. WS-BPEL is a workflow language for business processes but lacks support for high-order workflows. Finally, JDSL describes the requirements of individual computational jobs but does not address the runtime management of jobs or job relationships.

4.10 Software

The software can and the accompanying documentation may be obtained from the following svn server: https://scm.gforge.inria.fr/svn/grid4all/wp1/Jade and https://scm.gforge.inria.fr/svn/grid4all/wp2/DepOz.

4.11 Innovative usages

The VO management system is used to support creation and operation of VOs as illustrated by the following simple scenario: A VO member invokes the membership service through the web interface to log in to the VO and then contributes some computational and storage capacity to the VO through the basic resource manager (RM). Then, the user invokes the registry to lookup a service. If the service is unavailable, the user invokes the deployment service to deploy an application implementing the needed service. The ADL-based deployment service discovers and allocates resources to host the application, instantiates the components on these resources, binds them, and activates the application. The service is registered in the registry and can be accessed by the user.

DepOz is a programming framework used to express parameterised, dynamic Fractal-based architectures. Consider a tree-based, decentralised deployment process where each node of the tree locally deploys one instance of a component, initiates deployment on its branches, and synchronises on them. As explained in D2.2, few lines of DepOz code are sufficient to express this process. As another example, consider the following DepOz expression: *Sensors= {Deploy Hosts SensorPackage};* this expression captures the following semantics: deploy a sensor component (packaged as *SensorPackage*) on every host belonging to the set *Hosts* and collect the results in the set *Sensors*. When the value of *Hosts* changes (i.e., a new host is added/removed to/from the set), then automatically deploy/undeploy sensor components and update *Sensors*. As a final example, complex deployment workflows combining control and data flow patterns can be expressed in WorkflOz as simply as: *Task={Seq {MultiMerge {Sync Task1 Task2} Task3} Task4}*

4.12 Conclusions and lessons learnt

This chapter has presented the tools of the Grid4All VO management system and DepOz, an Oz-based framework for managing distributed component assemblies. Deployment is a central part of VO management, and our initial work had led to ADL-based deployment service, fully integrated with Niche (D1.5). ADL-based deployment is easy-to-use, but specifies only the initial deployment. Dynamic reconfiguration is enabled in conjunction with Niche that provides programmatic interfaces for application managers to dynamically reconfigure applications when unexpected events occur; e.g. provision a new component on failure of a node. This programming model is low-level, procedural and limits reuse.

This has lead to our work on languages that provide high-level support for developers to describe and specify declaratively application architectures exhibiting arbitrary workflows. With its higher-order and multi-paradigm features, Oz forms a powerful base for developing frameworks and domain-specific languages for deployment and configuration. Our current DepOz framework is a first iteration and we intend to extend it in many directions: (i) by defining patterns for self-configurable systems and capturing them in Oz libraries similar to WorkflOz, (ii) extending WorkflOz with support for transactions, (ii) integrating DepOz with Niche in order to exploit the scalability and robustness benefits provided by self-configuring overlays as well as to extend the Niche distributed component management platform with the high-level language support for self-deployment and dynamic reconfiguration.

5 Information Service

5.1 Overview and principles

Grid systems formed of distributed and heterogeneous resources and services should provide users and software means to discover resources and services. In the context of Grid4All, the Internet is a ubiquitous utility. Internet resources are traded in a market-oriented environment with auction mechanisms to arbitrate exchanges. Auctions are temporally and spatially confined. The Grid4All Semantic Information System (G4A-SIS) provides a registry, a matching and a ranking/selection mechanism to peers[4] (participants) offering or requesting resources. Resource characteristics and auction properties are used to satisfy peers' intentions and preferences. Peers discover trading instances, i.e. market services that trade matching orders.

The key innovation of the G4A-SIS is to bring the rich benefits of semantic-based discovery to a democratized grid environment, i.e., an environment where services are numerous; time-to-market is short and where the major stake is ease of use and simplicity for users and developers. Semantics enrich the discovery process since they are able to match peers that use different vocabularies. The need for semantic services in democratized settings and the benefits it brings are stated in 5.2. The sections that follow present technologies developed for realizing a Semantic Information System of an open Grid e-market environment.

5.2 Purpose and features offered

G4A-SIS gives users of the resource market place the ability to discover auction markets according to both the characteristics of the traded resources and also that of the auction. Furthermore G4A-SIS enables VO members to discover available services using semantically annotated descriptions. Semantic descriptions of entities (instances of an ontology) to be discovered are stored in the G4A-SIS registry. Java-based programming interfaces may then be used to query and retrieve information concerning advertised services from the system. The principal objectives of the Semantic Information System are to:

- Permit semantic and syntactic interoperability between peers with different conceptualizations of the traded resources/services. This is achieved through semantic annotation of descriptions of discoverable services. Semantic annotations help to resolve ambiguities between peers for the meaning of parts of services' descriptions;
- Facilitate automated registration and discovery of services within an open environment;
- Simplify usage through tools to translate WSDL specifications with textual annotations to formalized annotations. The system discovers mappings between WSDL specifications and ontology classes and generates OWL-S documents. Matchmaking of services is then performed at the semantic level (i.e. matching ontology classes);
- Provide open, extensible, and interoperable mechanisms imposing minimal requirements from the operating environment;
- Allow personalized selection based on user preferences and reputation.

G4A-SIS provides the following main functionalities:

- Advertisement: Advertising of market-related request or offer information related with traded resources and services, and
- Query: Querying in order to obtain a list of relevant services: application services or market services that expose characteristics and information about resources ordered (as a consumer request or as a seller offer) within these markets.

[4] We use the term peer to refer to software agents acting on behalf of participants within a democratic grid.

5.2.1 Market advertisement

Market advertisement is the process of inserting new resource descriptions as *offers* or *requests*. Offers and requests are entailed with description of the auction at which they are submitted. All descriptions (offer, request and market) are in the form of RDF triples, i.e., (subject, property and object). Advertisements are done using the G4A-SIS Web Service API that alleviates users of low-level formalities of ontology specifications.

Offers and requests may be simple or complex. Requests are exclusive disjunctions (XOR) of simple or conjunctions of orders. Such descriptions are mandatory to request bundles of resources; for example, consumers must be able to request for 8 CPUs and 240 GB of storage space atomically. Offers and requests are traded at auction-based markets and their descriptions contain the following information:

1 Description of market instances where the resource/service will be traded, including the location, the starting and closing times.

2 Description of characteristics of the traded services or resources in terms of capacity, quality of service, time of availability, etc.

3 Information concerning the required pricing policy and minimally (or maximum) acceptable prices.

4 Contact information of the provider or consumer that has initiated the market.

Both Offers and Requests are specific kinds of *Market Orders*.

5.2.2 Application Service advertisement

Service providers submit WSDL descriptions of the service along with an XML-schema based annotation document that provides textual annotations and/or a mapping between advertised service I/O types and concepts in OWL ontologies (semantic annotations). An *OWL-S profile* document is generated and inserted in the G4A-SIS registry. Referred OWL ontology elements must be registered at the G4A-SIS and be publicly available before inserting advertisements.

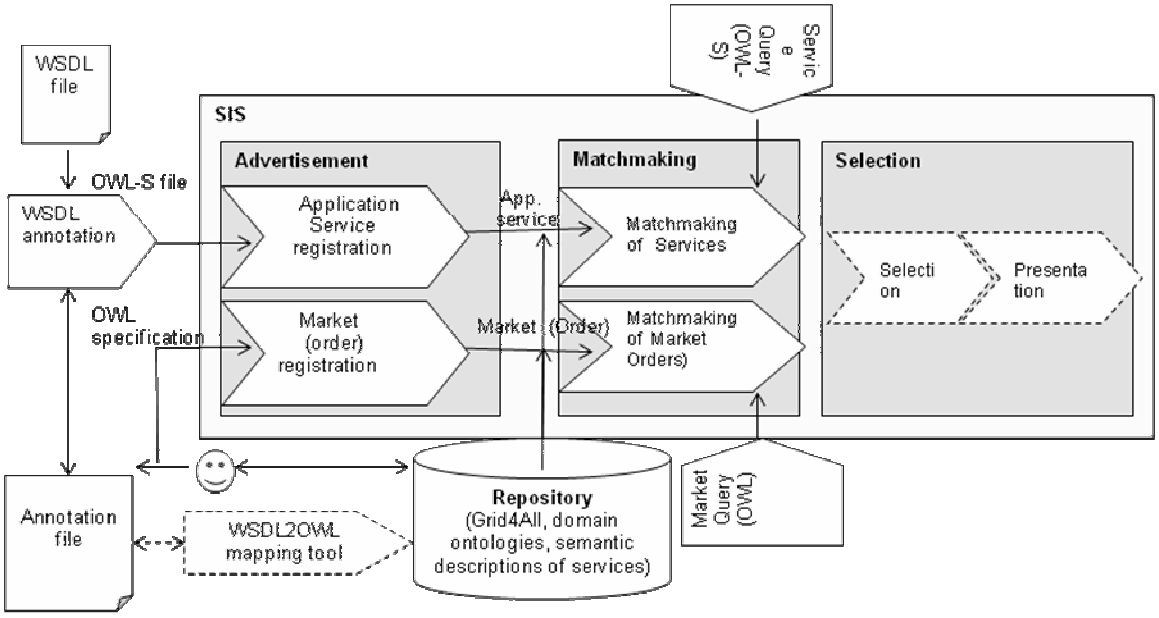
Before registering, the provided information is inspected to ensure that there are no type mismatches and that registered descriptions are consistent in the knowledge base.

5.2.3 Querying

The G4A-SIS may be queried to obtain ordered lists of available and published markets matching the required criteria: type and quantity of resources, QoS attributes and parameters of auctions at which required resources are traded. The G4A-SIS matches submitted offers and requests and presents a ranked selection of advertised markets. Queries are submitted using the G4A-SIS Java API.

5.3 G4A-SIS Architecture

The SIS high level architecture is depicted in the following diagram. This diagram also illustrates the flow of data for the main use cases (registration, advertisement and query).



The main components of G4A-SIS are:

- The application service registration module,
- The WSDL Annotation tool to attach semantics to service descriptions,
- Market order registration,
- Matchmaking of Services to filter services based on query criteria,
- Matchmaking of Market orders to filter registered orders based on query,
- Selection module that ranks the results of the matchmaker,
- Repository of semantically rich descriptions of markets and services,
- The Presentation layer that furnishes a Web interface to human agents.

5.4 Implementation

The matchmaking and selection components interact through an internal API and are accessible as a whole through a coherent external API available as a set of Web Service operations. The G4A-SIS may also be used as a standalone component, invoked locally by client applications through Java APIs. The rationales to provide a web service implementation are:

- Provide a well known access technology to G4A-SIS,
- Facilitate interoperation with other Grid4All components,
- Facilitate automaton of system testing and benchmarking.

5.4.1 Market matchmaking

This component performs automatic classification of individual markets by computing their inferred types described using the Grid4All ontology developed [Vouros et al, 2008] in OWL-DL. Clients send SPARQL queries to filter individuals matching specified order-related constraints (on offer or request) and market-related properties. The Pellet[5] reasoning engine is used to automatically classify individuals according to constraints in the queries.

[5] http://clarkparsia.com/pellet/features/

Request emitted by consumers and *Offer* emitted by providers are types of *Order*. Individuals of class Offer are subclasses of Offer and are matched and classified under specific request subclasses. Individuals of class Request (queries) are subclasses of Request and are matched and classified under specific offer subclasses. Individual auction-markets are classified as subclasses of Market class. To match offers and requests, the resource specification, the time specification and other properties related to the order and the market should match [Vouros et al, 2008].

5.4.2 Service matchmaking

OWL-S service profile descriptions generated by the automatic translation of WSDL specifications are stored in the G4A-SIS registry as advertisement instances. A list of input/output types is submitted in the form of a query to the G4A-SIS where the I/O types are expected to refer to existing ontology classes/individuals. For example, a query for services that has an input parameter of type “Compute Node”, and an output parameter of type “Hard Disk”, will have the form of an OWL-S profile document, in which there is an input parameter and an output parameter of the respective types.

The semantic matching is divided in two main stages: a) matching of inputs to ensure proper execution of service, and b) matching of outputs to fulfil the demands of the service requester. Three basic types of matching are defined in the context of Grid4All services: “Exact”, “Subsumes”, and “Fail”. Let T be the terminology of the domain ontology where the service I/O types are specified; CT_T the concept subsumption hierarchy of T. The types of service matching in the context of Grid4All are the following:

- Exact match. Service S exactly matches request R $\Leftrightarrow$ IN_S, IN_R: $IN_S \doteq IN_R$ $\Leftrightarrow$ OUT_R, OUT_S: $OUT_R \doteq OUT_S$. The service I/O signature perfectly matches with the request with respect to their formal semantics.
- “Subsumes” match. Request R subsumes service S $\Leftrightarrow$ IN_S, IN_R: $IN_R \sqsubseteq IN_S$ $\Leftrightarrow$ OUT_R, OUT_S: $OUT_S \sqsubseteq OUT_R$. This means that the advertised service might be invoked with a more specific input than expected and that the required service might receive a more specific output type than expected.
- Fail. Service S fails to match request R in any of the ways described above.

5.4.3 Service Selection

The Selection Service (SS) ranks and selects matching services and markets. It narrows the relevant providers computed by the Matchmaking Service (MS). The main method of this service is *selectProviders* that returns the list of providers according to consumer preferences such as reputation, provider preference against the service that is required to be satisfied and finally the provider load. Consumers and providers may set their preferences using the *setPreferencesForConsumers* and *setPreferencesForProviders* method to inform the G4A-SIS of their preferences.

The main innovation of the selection service is to balance both consumer preferences and provider preferences. For a given query q that should be satisfied by a provider p, the selection service considers both the preference of the provider to satisfy q and also the preference of the issuing consumer to select the provider to compute a score for each provider p. Let CI (PI) represent the consumer (provider) preference vector, the score of p is computed as follows:

$$scr_q(p) = \begin{cases} \left(\overrightarrow{PI}_q[p]\right)^{\omega}\left(\overrightarrow{CI}_q[p]\right)^{1-\omega} & if\ \overrightarrow{PI}_q[p] > 0 \wedge \\ & \wedge\ \overrightarrow{CI}_q[p] > 0 \\ -\left(\left(1-\overrightarrow{PI}_q[p]+\epsilon\right)^{\omega}\left(1-\overrightarrow{CI}_q[p]+\epsilon\right)^{1-\omega}\right) & else \end{cases}$$

The parameter ***ω*** ensures such a balance and takes its values in the interval of [0...1].

5.4.4 The G4A-SIS Registry

The G4A-SIS contains a registry that stores published markets and application-specific services. Services are represented as OWL-S documents (OWL-based upper level ontology of service concepts to describe properties and capabilities of Web Services). SIS uses the Service Profile to match queries to advertisements. The G4A-SIS in fact maintains two registries: The Resource Registry, which is essentially the Grid4All ontology, and the Service Registry which is a directory (in the file-system where the G4A-SIS resides) where service descriptions encoded in OWL-S documents are stored. A provider submits a service description in WSDL, together with textual annotations of its input/output parts. The output of the annotation process is a service signature specification in OWL-S, which is inserted in the G4A-SIS registry.

The management of G4A-SIS registry and the reasoning tasks are performed using the JENA[6] framework and the Pellet 2.0 reasoner APIs. These APIs operate at the level of generic RDF/OWL triples.

5.4.5 The Resource Registry

The Grid4All ontology consists of: a) Resources, b) Market orders (offers and requests), c) Markets, and d) Agents. Resources are *offered* by providers or *requested* by consumers and traded in e-markets. Resources are described at the logical level and may be Atomic, Aggregate or Composite. Currently only Compute Nodes and Clusters may be traded at Grid4All auction markets. Agents are providers or consumers. A provider offers tradable resources at markets and advertises these markets at the G4A-SIS. Consumer (providers) agents discover such provider (consumer) initiated markets by placing queries describing their requests (offers).

	Offer specified in PIM	Offer not specified in PIM	Request specified in CIM	Request not specified in CIM
Offer specified in PIM[7]	x	x	x	√
Offer not specified in PIM	x	x	√	x
Request specified in CIM[8]	x	√	x	x
Request not specified in CIM	√	x	x	x

Table 2 Possible match between Offers and Requests

A market order may be part of either an advertisement or a query for advertised markets but not of both. Table 2 shows the possible matches between markets and orders. G4A-SIS allows providers and consumers to discover endpoints to markets and select markets according to their requirements and preferences.

The query process involves insertion of new elements to the Grid4All ontology, reasoning with the ontology and performing SPARQL queries. The ontology is engineered so that reasoning mechanisms suffice for matchmaking. To take advantage of the classification mechanism, and according to the requirements for offers and requests, market orders are represented as follows: Resources' offers are represented as individuals of class Offer and requests are represented as defined classes. Generally, abstract descriptions are represented as defined classes, while concrete descriptions are represented as primitive classes and/or individuals. The inference engine is used to find the inferred types of the individuals, or inferred super-concepts and sub-concepts. Potential query results can be obtained this way, but since defined classes

[6] http://jena.sourceforge.net

[7] Provider Initiated Market

[8] Consumer Initiated Market

cannot represent all the constraints posed by agents, additional SPARQL queries need to be created, in which these constraints can be expressed, and executed on the ontology. The following algorithm describes the main steps following submission of a request for resources. Similar steps are followed for resource offers with the only difference that a search is done for classes (i.e. requests) under which specific individuals can be classified. At every request:

- Create and add a new class in the registry where the created class is defined using the restrictions on the properties as specified in the request
- For every request, create a SPARQL query to retrieve markets using the property restrictions of classes created in previous steps. If defined classes have been created in step 1, reference them in the SPARQL query: Only individuals that are classified as instances of these defined classes will be retuned as valid query results. If no defined class is created (i.e., there are no object property restrictions), reference the default primitive classes of the Grid4All ontology (Request, Resource, Tradeable_Resource, etc) in the SPARQL query.
- For every datatype property restriction, add a corresponding constraint to the SPARQL query. If the restriction is a single specific value, then add an equality constraint. If the restriction is a range of values, then add a "greater than" and/or a "less than" constraint.
- Execute the SPARQL query on the inferred model, using both the asserted and the deduced knowledge to retrieve results.

5.4.6 Service Registry

Semantic descriptions of advertised Web Services are stored in G4A-SIS as OWL-S documents. Software agents should provide service descriptions using WSDL documents. Human creators may browse available domain ontologies and associate the best fitting ontology concepts to service I/O types. Creation of semantic descriptions (OWL-S profiles), should conform to the following requirements:

- WSDL service descriptions typically generated from web service frameworks using source code of the services should not require modifications or interventions.
- Ontology concepts should be available to web service creators.

G4A-SIS uses additional XML documents with simple annotation blocks, one for each I/O parameter defined in the WSDL description. Information may be either natural language descriptions or comments or references to domain ontology concepts. The following figure shows an example of an annotation document.

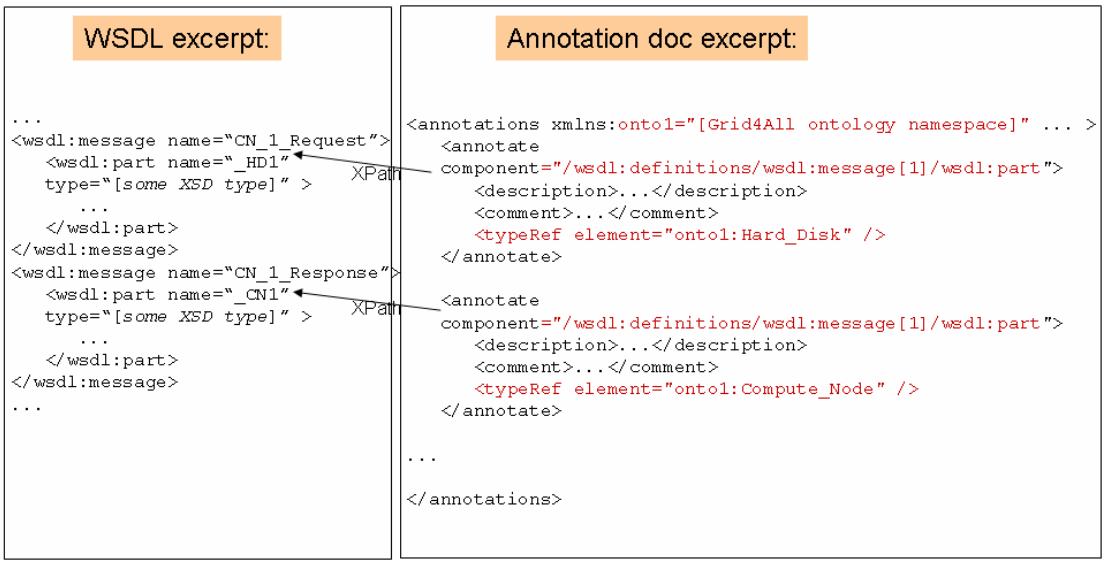


Table 3 Sample of an annotation document

Annotation blocks (starting with <annotate> tag) are related to WSDL message parts through XPath expressions; the original WSDL is not modified. Agents advertising applications services should provide the WSDL description and its associated annotation document. We provide a tool to parse the WSDL file and generate its annotation file with empty fields; service providers should add the content by filling the empty fields (concerning the WSDL I/O parameters) by consulting appropriate domain ontology. Information about the domain ontologies supported by the G4A-SIS can be obtained using appropriate external API calls. The WSDL and the associated annotation document when submitted to the G4A-SIS are translated to an OWL-S description using the mapped OWL classes of the domain ontology as described in [Vouros et al, 2008b].

5.4.7 Querying

Querying for services should associate service I/O types to lists of domain ontology concepts. Matchmaking finds advertised services with I/O types that either match the given I/O types or subsume them. Given that the agent making the query has *n* input parameters of types I_1, I_2, …, I_n, and *m* output parameters of types, O_1, O_2, …, O_m, exact matching is performed by finding all services which have input (output) parameters whose types match exactly with a parameter types defined in the service query.

The computed results returned to the querying agent are ranked, i.e. given a real value in the range [0 ... 1]. This functionality is provided by another method, namely *ServiceRanker*. The rank of a matched service, S, with respect to a service query, R, is equal to $aI_m + bO_m$, where a + b = 1. I_m is the semantic similarity between the set of inputs for S (I_S = {I_{S1}, I_{S2}… I_{Sn}}), and the set of inputs for R (I_R = {I_{R1}, I_{R2}… I_{Rn}}). I_m is equal to $\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} nosm(I_{Si}, I_{Rj})$, where *nosm( )* computes the semantic similarity between atomic concepts. O_m is computed in the same way using the output types. The similarity of atomic concepts is given by their distance in the domain ontology graph, and adjusted according to the type of matching (“Exact”, “subsumes”), so that exactly matching services obtain a higher rank. The Semantic Matching Framework (SemMF)[9] is used to compute the similarity between concepts.

5.5 Configuration

G4A-SIS is deployed as a web archive (.war) package and has been tested with Tomcat Servlet container that provides SOAP support. Stub client code in Java for invoking the service can be generated using the WSDL2Java axis tool with the G4A-SIS WSDL description as input. The G4A-SIS can also be used as a standalone library invoked locally by its potential client, e.g. the Grid Marketplace component. Detailed information on the installation and usage of the G4A-SIS can be found in the documentation of the system.

5.6 Used technologies

SIS is implemented using Java Enterprise technologies. The Jena framework and Pellet reasoning engine are used to implement matchmaking functions. Jena provides classes and interfaces corresponding to every concept of RDF(S) and OWL languages. Jena uses a simple SPARQL engine, ARQ, through which SPARQL queries can be executed during the matchmaking process. Finally, Jena offers the capability to attach inference engines, such as Pellet, to the models (the knowledge base). When an inference engine is attached to a model, query processing is enhanced since inferred statements, which may provide answers to queries, are discovered. Such an enhancement, of course, comes at the cost of query processing time.

A MySQL database (for persistent storage of RDF/OWL models) is used to store ontologies. Semantic descriptions of advertisements are stored in the database and if necessary reasoned with the updated ontology to obtain inferred statements on the newly registered entries. Inferred knowledge is cached for future use. Caching of inferred statements improves responsiveness of query processing since the classification is executed at object registration and not during query processing.

[9] http://semmf.ag-nbi.de/doc/index.html

The implemented system has been tested using the Apache Tomcat server. The AXIS 1.4 Web services framework is used to provide WS access to the G4A-SIS API. The G4A-SIS API, supported by AXIS, conforms to the WSDL 1.1 specification. Thus, the overloading of operations of the API is supported.

The Mindswap OWL-S API provided with the OWL-S API implementation is used to translate from WSDL to OWL. The WSDL specification of the G4A-SIS service can be obtained from the following URL: http://icsd-ai-lab.aegean.gr:8080/grid4all_sis/ws/SIS?wsdl.

5.7 Other related G4A components

G4A-SIS is invoked by market negotiator agents in the Grid Marketplace (GRIMP).

5.8 Current status

All main features described in this section are implemented and evaluated. The quantitative evaluation results are presented in the deliverable D5.3.

5.9 State of art comparable products

Specific requirements of the Grid4All project that have driven the design of the Semantic Information System are enumerated below. Major available software solutions have been evaluated against these criteria and a summary of result is presented in this section.

A. Grid resources advertisement and discovery.

B. Services advertisement and semantic discovery.

C. **Traded Grid resources semantically discovered in e-markets.** Semantic matchmaking of offers and requests that of traded resources in advertised markets.

D. **Selection:** Selection and ranking of services to allow agents to distinguish among a multitude of services claiming to fulfill her needs.

E. **Automatic semantic annotation**; Facilitate reuse of existing WSDL registries by providing automatic ways to semantically annotate Web Service descriptions.

F. **API support**: A semantic information system should itself be accessed as a service in order to allow software agents to register and request services and resources programmatically without needing to know details of the underlying semantic formalisms for service descriptions. In the context of Grid4All, this requirement implies the development of an API to provide the functionality supported at a level that will guide agents to interact with it, thus supporting the interoperation with other Grid4All components. Such an API must provide operations for the following, at least:

F.1. Registration of agents,

F.2. Registration of offers/requests and services using the ontology and OWL-S specifications,

F.3. Query execution (i.e. discovery of the matching resources/services,

F.4. Specification of peers' preferences,

F.5. Ranking of query results based on stated preferences,

G. **Scalability and performance**:

The following systems have been reviewed and compared with G4A-SIS:

- MDS4 -Globus Alliance (dev.globus.org/wiki/MDS4)
- Arc (http://www.nordugrid.org/)
- K-Wf Grid (http://www.kwfgrid.eu/)
- InteliGrid/SRMS (http://inteligrid.eu-project.info/)

- Atlas/OntoKit (http://www.ontogrid.net/ontogrid/index.html)
- SGM: Semantic Grid MatchMaker (http://linkinghub.elsevier.com/retrieve/pii/S1570826805000028)
- MyGrid/Feta (http://www.mygrid.org.uk/tools/service-management/feta/)
- Meteor-S (http://lsdis.cs.uga.edu/projects/meteor-s/index.php?page=0)
- ERGOT (www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0177.pdf)
- EGEE/ActOn (www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0111.pdf)
- BondGrid (www.waset.org/pwaset/v1/v1-25.pdf)
- Lamparter & Schnizler (2006)
- Ragone et al (2007)

As shown in Table 4, most systems do not provide support for a Grid Economy and very few provide automatic (or semi) semantic annotation of services. We nevertheless agree that other functionality available could be extended and re-used in Grid4All.

Criterion / **Approach**	(a)	(b)	(c)	(d)	(e)	(f)	(g)
GT4/Mds4	✓	No	No	No	No	Partially	✓
ARC	✓	No	No	No	No	Partially	✓
K-Wf Grid	✓	✓	No	No	Semi-auto	Partially	✓
InteliGrid/SRMS	✓	✓	No	No	No	Partially	✓
Atlas/OntoKit	✓	✓	No	No	No	Partially	✓
SGM	✓	✓	No	✓	No	No	N/A info
myGrid/Feta	✓	✓	No	No	No	Partially	✓
Meteor-S	✓	✓	No	N/A info	✓	Partially	N/A info
Ergot	✓	✓	No	✓	No	N/A info	N/A info
Egee/ActOn	✓	✓	No	✓	No	Partially	N/A info
BondGrid	✓	✓	No	✓	No	N/A info	N/A info
Lamparter & Schnizler (2006)	✓	✓	✓	N/A info	N/A info	N/A info	N/A info
Ragone et al, 2007	✓	✓	✓	✓	No	No	N/A info
G4A-SIS	✓	✓	✓	✓	✓	✓	✓

Table 4 Evaluation of state-of-art Grid information systems

5.9.1 Related work on discovering and selecting traded resources and services in Grid e-markets

Lamparter & Schnizler (2006) present an ontology-driven market to trade Semantic Web Services. They report an auction schema enriched by components to enable semantic-based matching and price-based allocation. This work merges classical auction algorithms with semantic matching capabilities. They propose

a communication language that formalizes orders (requests and offers) and agreements. This work contributes towards the Grid economy using a single type of auction: It does not deal with multiple types of markets and their distinguishing features. The matchmaking process does not use preferences of providers and consumers.

Close to our approach, the work reported in (Ragone et al., 2007), applies semantic matchmaking in P2P e-markets. It mixes in a formal and principled way the semantic expressiveness of DLR-Lite logic programs, fuzzy logic and utility theory to find the most promising partners in a peer-to-peer e-market. It considers peers' preferences and utilities in the matchmaking process. Logical specifications are modelled as soft-constraints (requirements and preferences on the peers' orders) using fuzzy logic. Focusing on knowledge representation, the framework offers an alternative to our work on formalizing the retrieval of resources in a Grid-related market-oriented environment: However the proposed framework is under implementation. Concerning the conceptualization proposed, although the matchmaking approach is market-oriented, it seems that the focus, as far as markets are concerned, is only on pricing (value of goods), i.e. on the process of retrieving resources based on price negotiations between peers. As a consequence, there is lack of information concerning market and order related properties that a peer may consider in order to trade a resource.

5.9.2 Discussions on State of Art and lessons learnt

We have reviewed comparable systems i.e. systems using semantic metadata to describe services. The Table 4 provides a fulfillment matrix, indicating the Grid4All requirements supported by these systems. No approach fulfills all Grid4all requirements.

Only Lamparter & Schnizler (2006) and Ragone et al (2007) provide specific support to trade resources in e-market places, but none of the systems meet the first three requirements at the same time (i.e. matchmaking of resources, semantic matchmaking of services, semantic matchmaking of traded resources/services ordered in Grid markets). Moreover, none of the related approaches provide APIs to support an integrated information service process. Finally, other from Meteor-S (Kunal et al, 2005) and K-Wf Grid (Babik and Hluchy, 2006) all provide automatic annotation of Web services and support re-use of WSDL registries (WSDL to OWL-S automatic translation) or automating SAWSDL specification of its model references.

It can be conjectured that a hybrid (implementation) approach using and extending existing functionality/services/sub-systems could have been satisfactory. The G4A-SIS was designed to meet requirements and its implementation conforms to the latest Semantic Web Services standards. Existing technologies and tools are reused (e.g. grid resource representations, a generic OWL-S matchmaking algorithm, WSDL2OWL-S tool) where possible. G4A-SIS fully complies with all requirements but one, i.e., scalability. Query-processing is a bottleneck with the centralized G4A-SIS. While we foresaw this issue, quantitative evaluation (reported in D5.3) confirms our intuition. This has been our main focus of research in the last period and our conclusive design in presented in the next sections.

5.10 Software

The G4A-SIS software can be downloaded as a web archive (.war) package and needed libraries from the following URL: http://icsd-ailab.aegean.gr:8080/grid4all_sis/downloads/grid4all_sis.war. General information about Grid4All and G4A-SIS and descriptions of the exposed Web Service operations may be found at: http://icsd-ai-lab:8080/grid4all_sis.

5.11 Design of a Distributed Information Service

This section presents a decentralized design of the G4A-SIS that improves the scalability and leverages the distributed Grid4All environment.

5.11.1 State of the Art

Distributed reasoning

This section discusses distributed algorithms for reasoning on modular ontologies. Modular ontologies are those partitioned into multiple autonomous and possibly distributed components. Every local ontology module is associated with a local reasoning engine that communicates asynchronously with its peers. They may have mutual/cyclic reuse of terms.

Bao [Bao et al, 2006] proposes distributed reasoning with localized package-based Description Logics (P-DL) semantics, allowing local modules to coordinate asynchronously with peers. A package is a fragment of an ontology assigned to a node in a distributed environment. Reasoning may be done with both inter-module subsumption and inter-module role relations. Bao allows arbitrary references of concepts among ontology modules but packages may be extended only with imports (from package to package) of concept names. The tableau algorithm for *ALC* reasoning is extended to an incremental distributed algorithm where each generated fact is sent to its destination ABox where clashes are detected. A destination of a fact is either its home package or the ABox tree in which it is (directly or indirectly) generated.

Although this approach alleviates known limitations such as inter-module subsumption and transitive subsumption propagation between modules and supports message passing between peer reasoning engines, it is based on strong assumptions on the shared and agreed conceptualizations among peers and dynamic nature of peers is not taken into account. Finally, the asynchronous function of local reasoners makes them depart from depth-first searching, affecting the space complexity of the algorithm.

Adjiman [Adjiman et al, 2006] propose inference in peer-to-peer settings where each peer answers queries by reasoning from its local theory and querying other semantically related peers (with who it shares part of its vocabulary). A peer, if it can not solve a consequence finding task locally, forwards it to its acquaintances. The **DeCA**, Decentralized Consequence finding Algorithm is applied to the **SomeWhere** semantic peer to peer data management system to translate data models to propositional logic. The restriction to propositional theories (OWL PL) limits the expressivity of the model and "reasoning" is translated to finding consequences via the calculation of (proper) resolvents (set of clauses obtained from resolution). Mapping between local theories is expressive, and queries can be translated only to maximal conjunctive (conjunctions of extensional - atomic - classes) rewritings suitable for aggregating the results.

Drago [Serafini & Tamilin, 2005] proposes reasoning with multiple ontologies interrelated by semantic mappings. It uses a distributed reasoning approach in which reasoning is the result of combination via mappings of local reasoning chunks performed in single local ontologies. It presents a sound and complete distributed tableaux-based algorithm. We have used this approach as the basis for the design of the distributed Semantic Information System and describe it in detail in later sections of this document.

[Stuckenschmidt, 2001] addresses reusing ontologies in contexts (contexts of use, and/or applications that exploit these ontologies) where only parts of the encoded aspects are relevant. It defines a viewpoint of ontology as a subset of the complete representation vocabulary relevant to a specific context. It introduces approximate subsumption operators that use relevant parts of the definitions in the ontology. Peers may have different viewpoint of the ontology of its acquaintances (due to the lack of mappings for some ontology aspects, or to a different conceptualization of a domain, or to a different context of exploiting the ontology) and perform approximate reasoning by exploiting peer descriptions in conjunction with its own. With this approach maximal subsumption vocabularies must be pre-computed: This is neither efficient nor feasible in dynamic settings.

[Soma and Prasanna, 2008] parallelizes the inference process for OWL knowledge-bases by examining partitions, to minimize workload duplication and data communication. Local reasoning is performed by a rule-based engine. Ontology is compiled into a set of rules using negation free Datalog semantics and the initial knowledge base is partitioned amongst the nodes. Two schemes are examined: data and rule partitioning. Distributed queries are preformed in rounds. In each round, every processor (node) applies the rule-base to

the data set to obtain inferred tuples. Each inferred tuple is sent to all other nodes where it can be used. Once the messages from all processors are received, the next round begins with the new tuples as input and the process is repeated. The algorithm terminates when each processor finishes a round without generating new tuples. No proof of soundness or completeness of the algorithm is provided. For data partitioning, the algorithm gives almost linear speedups with number of nodes with exception for hash based partitioning that generates random sets of duplicating nodes.

Data Layer

Distribution and decentralization in the data layer designs efficient and scalable storage and retrieval of data in peer to peer environments. Data is stored as RDF triples using semantic overlay networks using techniques such as Distributed Hash Tables (DHT).

Piazza [Tatarinov et al, 2003] proposes the peer data management system (PDMS), an extensible architecture where any user may contribute new data, schema information or even mappings between peer schemas. Using intra-peer and inter-peer mappings, a query reformulation mechanism resolves conjunctive queries. Each peer defines its own relational peer schema called peer relations; a query in PDMS is posed over the relations on a specific peer schema. It is assumed that relation and attribute names are unique to each peer. Peers contribute data to the system in the form of stored relations (analogous to data sources in a data integration system): Queries in PDMS are reformulated strictly in terms of stored relations stored locally or at remote peers. Names of stored relations are assumed to be distinct from those of peer relations and query reformulation is done centrally (i.e. all mappings are stored in a central location). PDMS exploits semantic mappings between peer and stored relations.

Edutella [Nejdl et al, 2002] uses a distributed network of peer nodes to store RDF documents in a super-peer network topology to minimize message broadcasting. Each peer is attached to a super-peer, and super-peers form a backbone. Mediation and transformation rules support querying on data based on different RDF schemes distributed at peers. Query distribution uses routing indexes stored in super peers, that is, lists containing metadata information and pointers to attached peers and (neighboring) super peers indicating the direction where specific metadata (schemas) are kept. Super-peers manage the routing indices and determine where to forward queries. When a (super) peer joins or leaves, it informs the super peer network about its schema, properties, values and ranges. Given that the same schema, property or value may reside in more than one peer, query routing is optimized by dynamically creating routing indices through clustering by analyzing query frequencies. Different schemata in different peers are supported by a mediation mechanism that produces mappings between RDF properties of different schemata. Edutella tries to minimize the number of messages and not directly the query execution times.

GridVine [Aberer et al, 2004] describes an architecture that leverages the scalability offered by structured overlay networks to build interoperable, large-scale semantic overlay networks. It manages a logical layer over a semantic overlay and a physical layer, P-Grid using a DHT. To support a variety of schemas two approaches are supported: Schema inheritance and semantic gossiping. Schema inheritance allows deriving new classes and properties from existing ones, thus creating complex hierarchies of categories and classes from the most popular base schemas. Semantic gossiping aims at establishing global forms of agreement starting from a graph of purely local mappings among schemas. Different peers may annotate their data according to the same schema. A peer may create (either manually or automatically) a mapping between two schemas, thus creating a link between two semantic neighborhoods. Queries are forwarded by semantic gossiping; starting from a given semantic domain, the query is transformed and iteratively traverses other semantic domains following translations links until diverges (following syntactic and semantic similarity measures) from the original query. Classes and properties are encoding using RDF Schema. Queries are formed using RDQL and mappings between different peer ontologies are encoded in OWL. Statements are stored as RDF triples and refer to data items shared in the P-Grid infrastructure. For each triple, the subject, the predicate and the object are inserted separately on the DHT. Every class, representing schemata is a subclass of the P-Grid Data Item, a meta-schema class and every property is a sub-property of the P-Grid Data Item Property meta-class. Schemas are stored on the overlay with its peer code as a prefix. Queries are resolved by routing messages on the P-Grid overlay to the peer(s) responsible for storing the data item.

The whole process generates O (log (|Π|)) messages: O (log (|Π|)) messages to resolve the P-Grid entry plus one message for the answer.

GridVine has limited forms of inferences via the inheritance of properties and propagates translation links between schemata by gossiping. An advantage of this approach is that it addresses semantic heterogeneity of peers, but queries only retrieve data in terms of the abstractions that schemata offer.

5.11.2 Principles and requirements

The requirements and principles of the distributed G4A-SIS presented below:

Exploitation of Grid4All ontology: Orders, resources and markets are the main system entities. Exploitation of the Grid4All ontology calls for a distributed reasoning facility to classify abstract queries, to update the registry with new classes of advertisements/requests and to retrieve instances.

Advanced reasoning techniques for query-answering and matchmaking: Query answering in the centralized G4A-SIS requires the classification of abstract concepts and retrieval of all instances of the classified concepts matching query criteria (for multiple attributes and range values). For services, matchmaking infers subsumption relations between classes annotating service input and output parameters. A distributed design must offer scalable reasoning that tolerates peer failures.

Advanced techniques for the retrieval of data based on multi-attribute and range queries: Even though the conceptual layer evolves with new advertisements and requests, the number of instances added/deleted at the assertion layer is much larger than the number of concepts. Churn-resistant data management techniques are required to balance query processing load and maintain consistency at conceptual and assertional levels.

Fast concurrent updates/advertisements of (semantic) descriptions: Advertisements, queries and updates of advertised resources/services need to be efficient however large and complex the descriptions. This requires distributed reasoning that allows large degrees of concurrency whenever possible.

Scalability and extensibility: Scalability is a major issue due to the large number of peers that act concurrently, either as providers, as consumers, or as third parties in the Grid4All market place. The system should tolerate large numbers of concurrent updates and accesses even if new peers with distinct (conceptual, reasoning, data management) capabilities enter the network.

Tolerance to churn and resulting inconsistencies: Churn may cause inconsistencies at the conceptual and/or the data (assertional) levels. Simple replication of data is necessary but insufficient and. transactional data access is required to enforce strong data consistency.

Support for peer heterogeneity: The centralized G4A-SIS operates with the Grid4All ontology and this should be extended to support peers with different conceptualizations. System should manage mappings between elements of different ontologies and store/retrieve these mappings.

5.11.3 Design of the Distributed Semantic Information Service

To satisfy the above requirements we advocate distinct logical and data layers: This is typical in databases to address data independency and has correspondences with the terminological (schema, conceptual, T-Box) and instances (data, assertional, A-Box) layers in the semantic web, description logics and OWL. The logical layer manages ontology elements of specific conceptualizations, their properties and mappings: it supports operations to answer intentional queries (i.e. classification of semantic queries and of the related

aspects), subsumption checking, instances' realization and the computation of semantic mappings between conceptualizations. The data layer manages data storage, i.e. store and retrieve instances of concepts ensure their consistent replication.

To bridge the two layers, data (instances) must refer to the concepts they realize and concepts must be linked to their instances. The two layers separate operations on the schemata and on the data, providing clear roles to the peers; those responsible for operations at the conceptual level and those at the data level. A single peer may play both roles. In contrast to GridVine that also separates logical and data (physical) layers, we do not map operations on triples at the logical layer to operations at the data layer and thus avoid introduction of new name spaces and storing of schemata specifications as triples at the data layer. The advantages of this separation are (a) load may be balanced between peers in function of their capabilities and roles (b) distributed reasoning may be supported independently of the way data is stored, (c) data management operations may take advantage of state of the art techniques in P2P systems, independently of the design of the logical layer and how reasoning is performed, and (d) efficiency may be achieved by forwarding (parts of) queries to those peers capable of answering them. A peer that manages a specific part of the ontology is responsible for both the conceptual and the assertional parts of the knowledge base.

Logical Layer

In this layer, peers store the conceptual parts of ontologies in order to: to perform reasoning, to retrieve instances from the data layer and to aggregate the final answer to send to the requestor. There are many ways to design this layer: In peer to peer systems motivated by data management the logical layer adheres to a simple data model, usually the RDF/RDFS model that is stored as triples and is used to drive multi-attribute query-answering and range queries at the data layer. Here no inference is done (e.g. for answering queries in cases where not all the attributes have been specified, or queries in cases where terms used are at a different level of abstraction than those used for indexing data) since the semantics of specifications are not made clear. Furthermore, in structured overlay networks, triples (storing RDF/RDFS specifications) are usually indexed and stored three times; adding to the load of peers as well as to the redundancy of specifications, and thus to the difficulty of maintaining consistency of the stored data.

We cannot adopt this approach at the logical layer, since we need (a) to exploit specifications made in expressive models (OWL DL), (b) reasoning techniques to exploit such specifications, (c) reduce redundant indexing but introduce redundancy for availability and to render reasoning more tolerant to churn.

Motivated by the need to perform distributed reasoning in knowledge bases, some peer to peer systems use unstructured overlays and according to the peers' semantic similarity enforce distributed/parallel reasoning algorithms: The emphasis here is to bridge peers' conceptual specifications by providing mappings and to exploit these semantic mappings to propagate / expand reasoning to peers with knowledge of different aspects of a domain or that may have "something to add" to the conclusions (as already shown in 5.11.1).

Summarizing this discussion the main issues that we address in the design of the distributed G4A-SIS are: (a) The way specifications are modularized and "separated" to different peers, (b) how to bridge (map) the different specifications (c) how to contact local reasoning (depending on the language used) and propagate reasoning to related peers, (d) how to locate all peers that are relevant to a query, (e) how to cope with churn, i.e. peer leaves/joins (f) how to deal with different conceptualizations (i.e. peers heterogeneity), (g) how to provide extensional answers to queries effectively.

Modular Logical Layer: Modules, bridge rules and distributed reasoning

We modularize the Grid4All ontology and specify "bridge" rules between different modules to provide different peers with different responsibilities concerning the operation on elements at the conceptual and data layers by following the approach specified in [Serafini & Tamilin, 2005].

Each module is a description logic that is at most equivalent to $\mathcal{SHIQ}$ and contains concepts' and properties' descriptions, including general axioms (subsumption or equivalence) relating descriptions as well as

declarations such as those concerning inverse and transitive roles. Given a set of indexes *I* used to identify modules, let $\{M_i\}_{i\in I}$ be the collection of modules of an ontology. Every M_i formalizes a part of the ontology, both internal and external concept and properties' definitions. A module may be considered as an ontology *i*, focused on specific aspects of a domain. A concept *C* in module *i* is denoted as *i:C.* Similarly, the axiom *i:C* $\sqsubseteq$ *D* denotes the fact that the axiom *C* $\sqsubseteq$ *D* is in the *i-th* module. Such a specification resembles the internal concept definition in [Bao et al, 2006], where *C* is a concept name and *D* is a concept expression.

Mappings between modules are expressed as collections of bridge rules. A bridge rule may have one of the following forms:

$\sqsubseteq$ *i:A* *j:G* (onto-bridge rule)

$\sqsupseteq$ *i:A* *j:G* (into-bridge rule)

i:A $\equiv$ *j:G*

Where *A* and *G* are concepts in M_i and M_j respectively.

The mapping rules of the latter type provide external (to module M_i) concept definitions. In both of the first types, bridge rules from *i* to *j* express relations between *i* and *j* viewed from the subjective point of view of the j-th module. Hence, a modular ontology at the conceptual level *T* consists of a collection of modules (T-Boxes) and a collection of bridge rules between them. To define the semantics, each module is locally interpreted in its local domain. Thus, there is a family of interpretations $\{\mathcal{I}_i\}_{i\in I}$, one for each module M_i. Each $\mathcal{I}_i$ is the local interpretation and it consists of a *possibly empty* domain $\Delta^{\mathcal{I}i}$, and a valuation function $\bullet^{\mathcal{I}i}$ that maps every concept to a subset of $\Delta^{\mathcal{I}i}$ and every role to a subset of $\Delta^{\mathcal{I}i} \times \Delta^{\mathcal{I}i}$. To complete the specification of semantics we need a family of domain relations:

A domain relation r_{ij} *from* $\Delta^{\mathcal{I}i}$ to $\Delta^{\mathcal{I}j}$ is a subset of $\Delta^{\mathcal{I}i} \times \Delta^{\mathcal{I}j}$. We use $r_{ij}(d)$ to denote $\{d' \in \Delta^{\mathcal{I}j} \mid \langle d,d'\rangle \in r_{ij}\}$. A domain relation maps the elements of $\Delta^{\mathcal{I}i}$ into the domain $\Delta^{\mathcal{I}j}$, from the perspective of *j*. This type of mapping is particularly useful for the retrieval and assertion of instances at the data layer. A distributed interpretation $\mathcal{J} = \langle\{\mathcal{I}_i\}_{i\in I}, \{r_{ij}\}_{i\in I}\rangle$ of a modular ontology at the conceptual layer consists of local interpretations $\mathcal{I}_i$ for each M_i on local domains $\Delta^{\mathcal{I}i}$ and a family of domain relations r_{ij} between these local domains.

A distributed interpretation $\mathcal{J}$ satisfies the elements of a modular ontology according to the following clauses: for every *i,j*$\in$ *I,*

1. $\mathcal{J} \models$ *i:A* $\sqsubseteq$ *B if* $\mathcal{I}_i \models$ A $\sqsubseteq$ B

2. $\mathcal{J} \models M_i$ if $\mathcal{J} \models$ *i:A* $\sqsubseteq$ *B* for all *i:A* $\sqsubseteq$ *B* in M_i

3. $\mathcal{J} \models$ *i:A* $\sqsupseteq$ *j:B if* $r_{ij}(A^{\mathcal{I}i}) \subseteq B^{\mathcal{I}j}$

4. $\mathcal{J} \models$ *i:A* $\equiv$ *j:B if* $r_{ij}(A^{\mathcal{I}i}) = B^{\mathcal{I}j}$

5. $\mathcal{J} \models Bij$ if $\mathcal{J}$ satisfies all bridge rules from M_i to M_j, where *Bij* is the set of bridge rules from M_i to M_j.

6. $\mathcal{J} \models T$ if for every *i,j in I,* $\mathcal{J} \models M_i$ and $\mathcal{J} \models B_{ij}$.

T implies a specification, if every interpretation that satisfies *T* also satisfies this specification.

T (respectively, a concept *C* in a module M_i of T) is can be satisfied if there exists an interpretation $\mathcal{J}$ such that it satisfies *T (*respectively*,* it satisfies *T and C* $^{\mathcal{I}i} \neq \emptyset$). To decide whether $T \models$ *i:A* $\sqsubseteq$ *B* ,similarly to reasoning in description logics, we translate this into the problem of not finding a distributed interpretation $\mathcal{J}$ such that $(A \sqcap \neg B)^{\mathcal{I}i} \neq \emptyset$.

According to [Serafini & Tamilin, 2005], given an acyclic distributed and modular T-box we have a set of local reasoning procedures Tab_i, one for each *i*$\in$ *I*. Given a concept *C*, Tab_i returns a completion tree which is a finite representation of a local tableau for *C* in T_i. On top of each Tab_i there is a distributed tableau procedure dTab_i, one for each i$\in$ *I.* The distributed tableau algorithm checks whether there are bridge rules capable of closing a local tableau. If the distributed tableau fails to close the initial local tableau, then the algorithm returns "satisfiable", otherwise it returns "unsatisfiable". The algorithm is specified in detail in [Serafini & Tamilin, 2005] and briefly presented below:

To verify whether the concept *C* is satisfiable in a T-box T_j of acyclic distributed T-box *T*, the distributed algorithm proceeds as follows. First, it applies a local tableau procedure Tab_j in order to build a local completion tree. According to the tableau algorithm, each node *x* introduced during creation of the

completion tree is labeled with a function $L(x)$ containing concepts that x must satisfy. If there is a clash in the tree then we do nothing since the concept C is unsatisfiable locally, otherwise the algorithm traverses the nodes of open branches and further checks whether the branches can be closed by the bridge rules. For example, to check the node x the algorithm looks for the bridge rules that can potentially enrich $L(x)$ and cause a clash in x. If there are such bridge rules, they are verified by calling corresponding foreign distributed tableau procedures. The algorithm completes when all opened branches in the initial completion tree T are verified. The basic operator on which this reasoning process is based upon is the propagation operator:

Given a set of bridge rules B_{ij} between modules M_i and M_j, the propagation operator P_{ij} takes as input a T-box in M_i and produces a T-Box in M_j as follows:

$$P_{ij}(M_i) = \left\{ G \sqsubseteq \bigsqcup H_{k=1..n} \;\middle|\; \begin{array}{l} M_i \models A \sqsubseteq \bigsqcup B_{k,\,k=1\ldots n} \\ i{:}B_k \sqsubseteq j{:}H_k \text{ is a bridge rule} \\ i{:}A \sqsupseteq j{:}G \text{ is a bridge rule} \\ \text{for } k=1\ldots n,\ n\geq 0 \end{array} \right\}$$

The Modular Grid4All Conceptualization

Given the GridAll ontology described in [Vouros et al, 2008], and according to the definitions above, we distinguish the modules that are graphically depicted in Figure 4. This figure shows only the backbone of the ontology with only some of the subsumption relations between concepts and only some of their (most critical for reasoning purposes) properties. However, it clearly shows the concepts elaborated by each module and the relations between these four modules.

Given an order (offer or request), this is related to a specific resource and market. We refer to these relations as they play a critical role in advertisement and retrieval of resources. According to the proposed modularization shown in Figure 4, we need to specify bridge rules representing relations between the classes of the different modules that must be exploited to propagate reasoning between modules; i.e. when one asks for an order in M_2, reasoning must be propagated to M_1 concerning the resources related to the markets retrieved. The generic rule to specify bridge rules in this case is the following one:

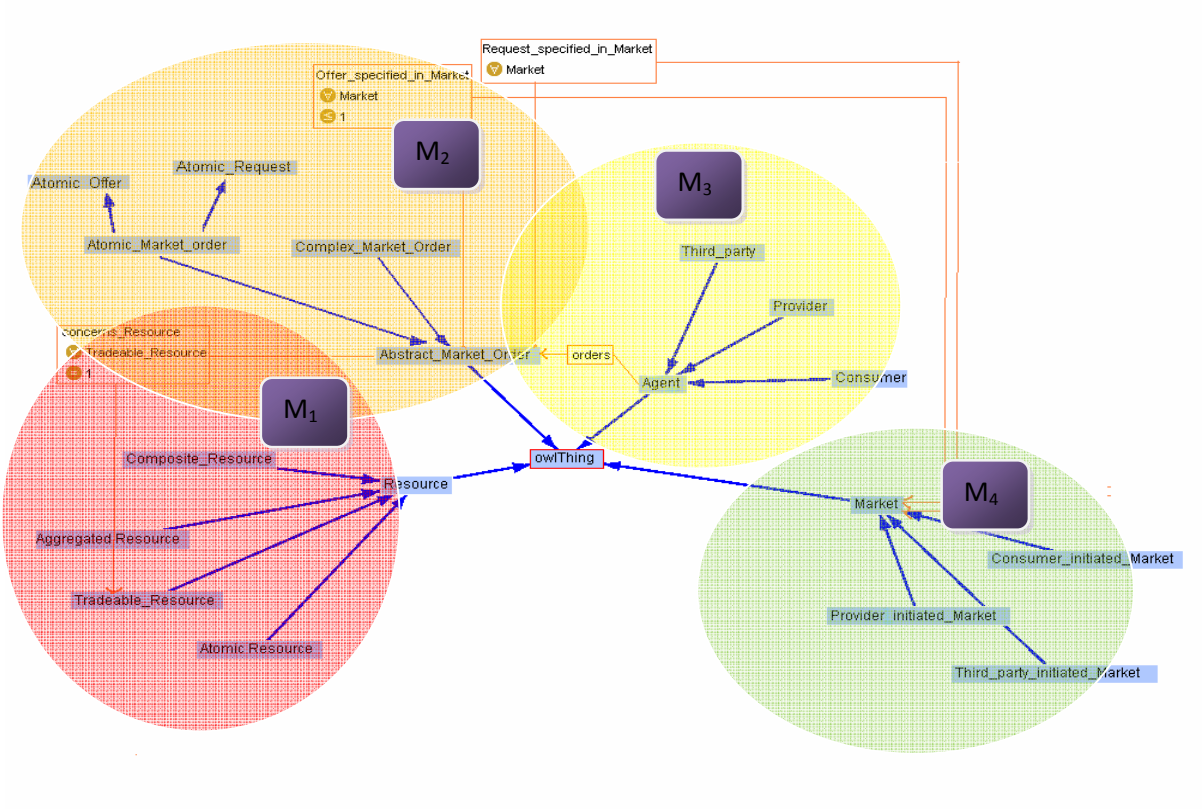


Figure 4 Modularisation of Grid4All Ontology

Given a property *P* with domain *i:C* (i.e. the concept *C* in module M_i) and range *j:D*, i.e. the concept *D* in module M_j, then

(a) Specify a new concept *virtual_D* in module *i*

(b) Specify that the property *P* in module M_i has range *virtual_D*, and

(c) Specify a bridge rule B_{ij} as follows: (i:*virtual_D* $\sqsupseteq$ *j:D*).

This last bridge rule permits propagating reasoning about subsumption relation concerning *virtual_D* in M_i to the concept *D* in M_j, and as will be described in the data layer allows propagating retrieval of instances of *virtual_D* in M_i to the retrieval of instances of *D* in M_j.

Design of the Logical Layer

For the semantic information system of Grid4All, dedicated to operate under specific assumptions of the Grid4All ontology (Figure 4), we propose the following design principles to structure the logical layer:

- All peers are aware of the whole Grid4All ontology, but, each peer is responsible to contact reasoning and maintain a specific module of the ontology.
- Modules are bridged by simple inclusion/equivalence mappings between their elements of the form (i:*A* $\sqsupseteq$ *j:B*), where i and j specify modules and A and B are classes of corresponding modules.
- Peers form groups responsible to contact reasoning and maintain data concerning a single module: I.e. peers in a group reason using the specifications in, and maintain, a specific module.
- Peers in a group (i) balance query processing load, (ii) are jointly responsible for updating their module (iii) each one provides a replica of their module and (iv) are jointly responsible for maintaining consistency among their replicas

- As the network of peers grows and new peers enter the network, there can be many groups for a single ontology module: Group members are responsible for maintaining consistency among group replicas.
- Groups elect representatives that are responsible to mediate intra and inter group communication for propagating reasoning and maintaining consistency among replicas at the conceptual and at the data layer: representatives can be re-elected in cases of failure or for load balancing purposes.
- Each peer joining the network joins a specific group of peers depending on (a) the load of the peers in the group, (b) the need to have more replicas in the group for increasing the reliability of the group, (c) the preference of the peer to contact reasoning for specific modules (e.g. due to being linked to specific providers, clients, or other peers with similar responsibilities), and finally to (d) the quality of mappings produced between the ontology of this peer and to other peers (we do not address this issue now: we shall elaborate on this later).
- Representative peers are coordinated through ad-hoc connections (i.e. by establishing connections among them via effective information sharing techniques), or via a management group that maintains a distributed registry of all representatives: The best alternative is subject to better study and experimentation, taking into account the dynamics of the network.

These principles result to structuring the logical layer as Figure 5 shows.

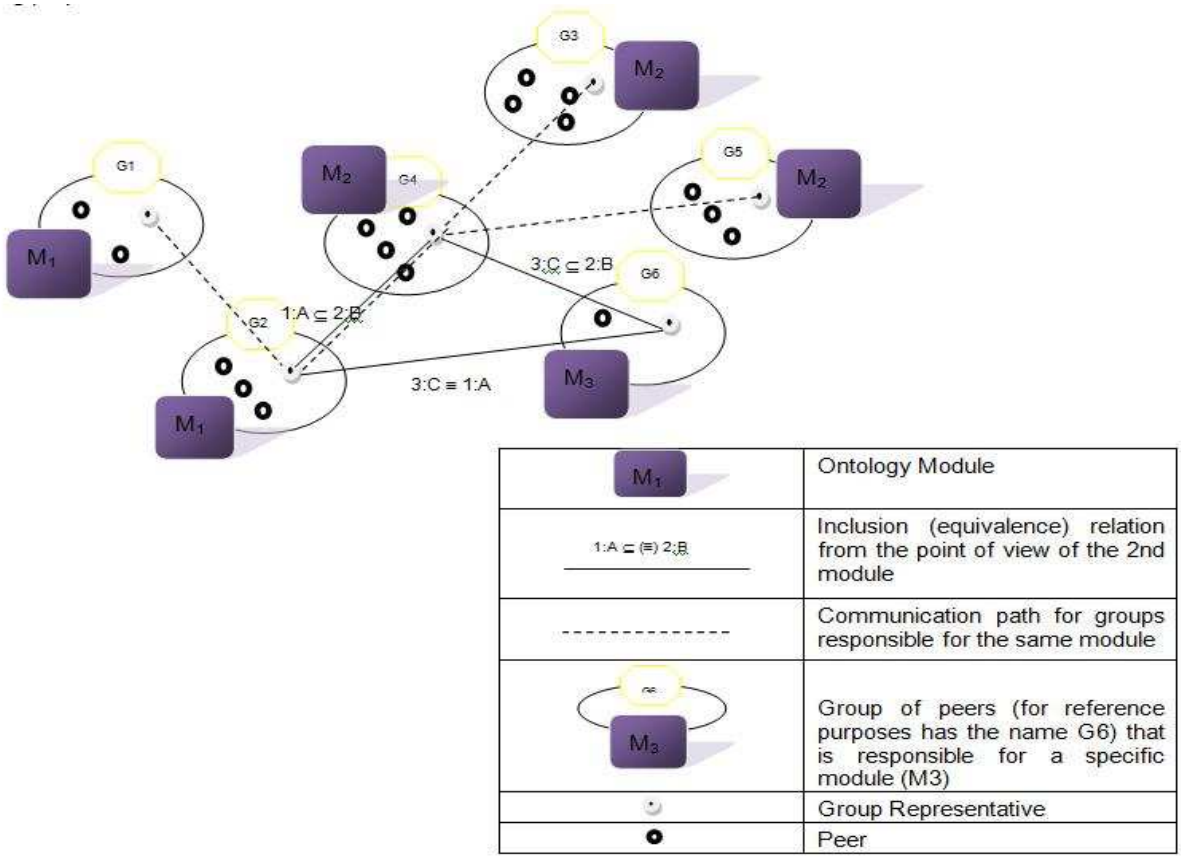


Figure 5 Peers organisation

As shown in Figure 5 each peer responsible for a specific ontology module is assigned to a peer group responsible for this module. As mentioned above, although each peer is responsible for reasoning and maintaining specific modules of ontology, it stores the whole ontology (which may not be updated from a specific time point). This allows peers to contact reasoning for any aspect of the ontology, while introducing redundancy leading to inconsistent ontology versions. Each group has one or more peers and a single

representative who is responsible to communicate with other representatives: (a) responsible for the same module and (b) responsible for modules related through inclusion/equivalence relations.

The first type of communication between groups allows to maintain consistency of replicas of the same module that are maintained by different groups, while the second type of communication allows peers to distribute reasoning, delegating reasoning tasks to other peers.

Such a structure of the logical layer ensures:

- ✓ Balancing load between peers in a group: Representatives, upon receiving a delegated task (e.g. to propagate reasoning and/or retrieve data) delegate this task to the least loaded peer.
- ✓ Tolerance to churn: Since any peer in a group can play the role of the representative, and any peer in the group can perform reasoning and maintenance tasks on behalf of the group. In the extreme case, a group may comprise of a single peer that performs all reasoning and ontology maintenance tasks as well as communication with other groups.
- ✓ Consistency of specifications among replicas of modules: Peers in a group that perform, or that become aware of, any action modifying the specifications in a module are responsible to enforce consistency within the group.

To assure consistency among the replicas of the ontology, two of possible alternatives are:

- All modifications to any part of the ontology are sent to all interested representatives, who will themselves maintain consistency within their group. For the distributed SIS, where the ontology will be composed of 3 or 4 modules, we expect that the number of representatives to be small compared to the number of peers in the system.
- A management group responsible for maintaining a registry of representatives. This group does not perform any reasoning or retrieval tasks. Their responsibility is (a) to forward queries to the representatives of other groups, and (b) to receive modifications for any module and disseminate these to the other groups in the system. This “central” group is not a “single point of failure” since it will contain more than one peer.

With this infrastructure, a request order or a service may be sent to any peer in the system: The receiving peer will forward the order to its representative who will find a group that can start to reason for this order. Starting means that a peer in this group may reason about this order and propagate reasoning to other modules, depending on the existence of bridge rules. As ontology modules change while the query is being propagated (as a consequence of distributed reasoning), representatives (as described above) become aware of these changes, and further update the replica of the ontology module for which they are responsible, propagating changes to other groups.

Heterogeneous Peers

Even if we consider modules as distinct ontologies and bridge rules as mappings between concepts of the distinct ontologies, heterogeneous peers (with different conceptualization) require more expressive mappings, involving properties, as well as instances, leading to cyclic mappings. To handle this, one way is to reduce the set of possible mappings, which is not realistic in real settings (to successfully reason with distinct ontologies, these need to be aligned taking into account the preferences of all peers involved).

An alternative is to reduce the scope of reasoning; peers with the same conceptualizations may form organizations, such as those described above (structured in groups depending on whether ontologies are modularized or not). The ontologies of peers belonging to different such organizations may be fully aligned. In this case, intentional query answering is performed within the scope of a single organization. Changes at the conceptual layer of an ontology may affect the set of mappings to/from other ontologies (maintained by different organizations), or changes may be propagated to the other ontologies (by addition/deletion of mapped elements). However these operations require further investigation and study.

Retrieval of instances, can also be performed within the scope of organizations of peers with the same conceptualization: A query may be answered “locally” by an organization and then be translated to the conceptualization (using the known mappings) of another organization, and so on. However, as instances are retrieved, to avoid duplicates, mappings between instances at the data layer need to be exploited.

5.11.4 Data Layer

The data layer is structured at different layers: At the “peers’ assertional layer”, at the replication layer, and at the transaction layer. Instances at the data layer are stored in peers’ assertional level of their knowledge bases (i.e. in A-Boxes). These instances are replicated at different peers in the group (or in groups related with the corresponding module), and peers via their representatives implement a strong data consistency protocol, to assure ACID properties.

Modular A-Box: Modules, bridge rules and reasoning

As shown above, knowledge is propagated via bridge rules to different modules (or ontologies) in the form of subsumption axioms. Here, we report on the consequences of bridge rules and individual correspondences (via domain relations) on the assertional (data) part of the distributed knowledge base. To express correspondences between semantically related heterogeneous individuals between A-boxes, we follow the approach described in [Serafini & Tamilin, 2005b] and introduce individual level bridge rules of the form $i:a \rightarrow j:b$ (individual correspondence). Such a bridge rule specifies that according to the point of view of *j* the individual *b* is *one of the possible* translations of the foreign individual a in the local domain of *j*. This means that it may also exist b' such that $i:a \rightarrow j:b'$.

A distributed A-box consists of a collection of A-boxes $A = \{A_i\}_{i \in I}$ and a collection of individual correspondences between pairs of corresponding A-boxes. Since the corresponding domains Δ_i can be heterogeneous, the semantic correspondences between heterogeneous domains are modeled using the domain relation r_{ij}.

In addition to the conceptual part presented previously and as far as the assertional part is concerned, a distributed interpretation $\mathcal{J}$ is said to satisfy the elements of the distributed A-Box if for every $i,j \in I$,

1. $\mathcal{J} \models i:x \rightarrow j:y$ if $\langle x^{\mathcal{J}_i}, y^{\mathcal{J}_j} \rangle \in r_{ij}$.

2. $\mathcal{I}_i \models C(a)$, $\mathcal{I}_i \models R(a,b)$ for all assertions C(a) and R(a,b) in A_i.

3. $\mathcal{J} \models A$ if for every i,j in I, $\mathcal{J} \models A$ and $\mathcal{J}$ satisfies all individual correspondences.

According to the semantics of the into-bridge rule $i:B \sqsubseteq j:H$, if in module M_i the concept *B* has an instance *b*, then in module M_j there should be an instance *h* of *H* such that the pair $(b^{\mathcal{J}_i}, h^{\mathcal{J}_j})$ belongs to the domain relation r_{ij}. Formally, if ($[i:B \sqsubseteq j:H]$ and $[i:b \rightarrow j:h]$), *then (if [i:B(b)] is satisfied , then [j:H(h)] is satisfied as well).* Instead of providing all correspondences, we may specify a transformation relation f_{ij} that allows transforming individuals from module M_i *to* individuals from module M_j such that these transformations respect the semantics of the domain relation r_{ij}.

Formally, if ($[i:B \sqsubseteq j:H]$), then *(*if *[i:B(b)]* is satisfied , then $[j:H(f_{ij}(b))]$ is satisfied as well*).*

The assertional propagation made according to the above patterns, ends in either (a) asserting new concept membership for existing individuals of target ontology when the first pattern is used, or (b) injecting new individuals to target ontology in accordance with the second pattern and further asserting their concept membership similarly to (a). In both cases, we will refer to such concept membership assertions as distributed concept membership.

The consequence of both propagational aspects affects distributed instance retrieval. The resulting instances can be naturally subdivided into two groups: ones that were explicitly defined in the target ontology A-box and the others that were injected as transformation of instances of source ontology. Therefore, according to [Serafini &Tamilin, 2005b], the distributed retrieval problem should be approached in a 3-step manner: (1) retrieve all local instances with respect to the distributed taxonomy of the target ontology, (2) retrieve all relevant instances of the source ontology and apply transformation to them, and finally (3) merge the results of two previous steps.

The Modular Grid4All assertional level

As Figure 5 shows, given an individual order in M_2, this is related to an individual market in M_4 and to an individual tradable resource in M_1.Therefore, by applying simple transformations to an individual order, we may locate the individual market and the individual resources to which this is related. We denote these transformations (with little abuse of notation) f_{OM} and f_{OR}, denoting transformations between concepts in

different modules rather than transformations between modules as shown above. Such a transformation, together with the corresponding bridge rules between corresponding concepts in modules allows injecting individual markets and tradable resources to the corresponding modules and concepts. Therefore, by exploiting the specified bridge rules an instance is classified under the most specific concept of the distributed T-Box.

For the retrieval of instances, we apply the algorithm specified in paragraph “Modular A-Box: Modules, bridge rules and reasoning”.

5.11.5 Design of the Data Layer

Each peer classifies each instance as an instance of a class of its own module, implementing local consistency. As this happens at the group level for the different peers in the group (each time), this assures also scalability. To avoid inconsistencies due to concurrent updates and/or churn, peers in a group need a local repair mechanism: a distributed consistency-preserving policy similar to the conceptual level. At assertion (or deletion) of an instance in a peer, the policy forces the peer to inform other peers in its group as well as other groups managing the same module.

Peers need to assure atomicity of transactions: This may be done at a group level (or at the level of many groups) using for instance the adapted Paxos commit protocol proposed in [Schütt et al, 2008b; Schütt et al, 2008b] to ensure consistency of replicated instances. Instances of different modules (e.g. a specific market with an order and a resource instance) may be connected via the above mentioned assertional propagation patterns.

To give a concrete example that requires connecting instances between modules let us consider the case of advertisements of resources (i.e. of their registration) and of their retrieval (i.e. of their discovery). Each offer made (advertisement) is related to an individual market place, and an individual resource (e.g. a cluster of compute nodes). To advertise this offer, the reasoning mechanism should:

- Classify the individual offer (o), market ($f_{Om}(o)$) and resource ($f_{OR}(o)$), each in the corresponding module, O, M, R.
- Create mappings of the form [O:o $\rightarrow$M: $f_{OM}(o)$] (f_{OM} maps an order to the related market instance) and [O:o $\rightarrow$R: $f_{OR}(o)$] (in this case f_{OR} maps an order to the related resource instance).

In the case of a request, which is a class related to market properties (m), order properties (o) and a class of resources (r), the reasoning has to do the following to retrieve the related advertisements:

- Classify classes order (o), market (m) and resource (r), in the corresponding module, O, M and R.
- Apply SPARQL queries to filter the instances related of the above classified classes.
- Synthesize the individuals retrieved into triples (i_o, i_m, i_r), such that it holds that [O:o $\rightarrow$R: $f_{OR}(o)$] and [O:o $\rightarrow$M: $f_{OM}(o)$].

5.11.6 Distributed Registration and Retrieval of Application Services

When querying services at the centralized SIS, the service inputs and outputs should be specified by providing lists of domain ontology concepts. Matchmaking is then performed to find advertised services with I/O types that either match the given I/O types, or subsume them.

Three basic types of matching are defined for Grid4All services: Exact match, “Subsumes” match, and Fail. Let *T* be the terminology (T-Box) of the domain ontology where the service I/O types are specified. There are described in section 5.4.2. To advertise and retrieve service descriptions the following assumptions are made:

A. Peers are homogeneous as far as the conceptualization of domains is concerned.

B. All services are indexed by means of the concepts that semantically annotate their input and output parameters.

C. Domain ontologies for indexing services are known to the peers who may form structured organizations to reason at the conceptual way in a distributed way.

D. In case of different conceptualizations for the same domain, then heterogeneity of peers is being treated.

Structured peer-to-peer networks performing "lightweight" subsumption checking [Skoutas et al, 2008] are suitable for distributed registration and retrieval of application services. We propose an approach that is well aligned with that to retrieve markets and resource. Moreover we take into account the cases where different conceptualizations for the same domain may occur.

5.12 Innovative usages

5.12.1 Formal conceptualization and Ontology

G4A-SIS provides an external traditional notion of resource and service matchmaking to the process of discovering auction-based markets that trades in such resources, and specific support for composite resources, i.e., when consumers (and providers) request (or offer) more than one type of resource and eases allocation of complex resource requests. In an open resource market place, multiple mechanisms may be used simultaneously. Using G4A-SIS, consumers and providers may also restrict their queries according to the auction mechanism that is employed. This is mandatory since consumer and provider agents may not be doted with capacity to participate at any type of auction.

5.12.2 Automating of semantic annotation process

Simplification of annotation process reduces design and development time for (human) service developers. Automatic annotation tools enable usage of semantic information services, which improves the discovery process but at the same time reduces complexity of usage for service developers. The semantic annotation maintenance process is also simplified. In contrast to other methods, our method of semantically annotating the input/output elements of Web Service descriptions allows exploitation of service contents in conjunction with textual descriptions. The accuracy of the semantic annotations computed by the USDS method proposed in the context of the Grid4All project [Vouros et al, 2008b] is quite high (above 0.78 for 7 experimental domains and almost 250 services). Conclusively this method reduces errors in annotation and consequently improves usability.

5.12.3 Ranking of matched resources and services

Matched queries are ranked by using preferences of both consumers and providers. A model to define participant perception of the system has been designed and metrics to evaluate the quality of query allocation has been proposed. The Satisfaction-based Query Load Balancing (SQLB) framework has been designed to satisfy both provider and consumer preferences and interests.

In large-scale information systems, query allocation is a challenge. Often queries are distributed amongst providers in a way to maximize overall performance, i.e., mainly balance load across providers. Our method preserve consumer interests as well.

5.13 Conclusions and lessons learnt

In the context of Grid4All a Grid market and service registry/discovery system was developed utilizing Semantic web technologies. The Semantic Information System is available through an API which hides the underlying ontological descriptions involved in advertisements and queries of available resources, services and markets, thus making the system usable by developer's agnostic to Semantic Web technologies.

SIS has been evaluated with some test data for performance reasons. It does not scale well for complex queries and large number of offered resources/services (more than 500) due to the non-flat and heavily axiomatized ontology (based on projects' requirements). Although G4A-SIS has many advantages over state-of-the-art systems (grid economy features considered in the semantic matchmaking process, automatic annotation of services, a selection functionality is provided, an API has been developed, etc.) it is restricted in terms of the provided architecture which is a centralized one (central semantic registry). For this reason, a decentralized version is envisaged. The design of a decentralized G4A-SIS is presented in this document, while the implementation is out of the projects' scope.

6 Market-based resource allocation

6.1 Overview and principles

With advent of cloud computing, i.e., virtualisation coupled with usage-based pricing models, organisations may scale their IT systems up and down and pay only for computing and storage capacity that they consume. The main cloud providers to cite are Amazon, IBM, Sun and Google, typically large companies basing their profits on the economies of scale. Such companies leverage their proprietary software and/or their in-house platforms to lease computing power as multi-tenant platforms. Cloud providers apply varying degrees of on-demand pricing to allow users to purchase the IT capacity that they require.

The vision of the Grid4All project is to democratize grid technologies and enable communities to create and deploy their own sharing infrastructures. The stake-holders (end users) that we address are individual residential users, small organisations either profit (small enterprises, start-ups) or non-profit (such as schools, educational organisations and associations).

Other Grid4All deliverables have described key middleware, services and tools for democratic grids developed for this purpose in this project. Chapters 4 and 5 of this deliverable have described some key resource management tools and services. This chapter describes the architecture and main software tools to enable providers and consumers to meet and trade compute resources at open market places. Trading resources can be assimilated to the function of resource allocation, i.e. decision of which resource is allocated to which user (or application) and when. When total demand exceeds the available capacity, requests may be rejected using different policies such as first-come-first-serve, by priority etc. Using market mechanisms decision to allocate resources is based on maximization of overall social welfare.

Different types of ICT resources may be traded (a) SMP-based servers (b) clusters with special interconnects (c) servers, desktops, portable computers and even specialized network attached storage. In Grid4All, we focus on the final category. Virtual machines have become the prevalent unit of deployment and render the underlying hardware platform transparent. The market place itself is insensitive to this change in unit of trade.

Considering the diversity of cloud vendors, the diversity of prices practised and services offered by cloud providers, automated market places for IT resources and services will find their place in Cloud computing. In a hybrid model where clients own some services and resources they could acquire surplus resources and services through the market place.

6.2 Purpose and features offered

The Grid4All Market Place (GRIMP) allows end users, i.e. resource providers to lease their idling compute resources, and resource consumers to buy time-limited leases. The market place provides an environment where interactions between these actors are formalized and where mechanisms establishing prices that balance supply and demand are employed. To argue why market places may be needed we present different alternatives. Organisations, companies and individuals may:

1. Invest in IT infrastructure either sized to cover peak loads or to cover average or median loads.
2. Completely outsource their IT needs to Cloud providers and run only operational expenses.
3. Mix the above two, i.e. invest in their own resources to cover average usage and rely on Cloud providers during peak periods.
4. As a variation of the third, consumers may contract minimum guaranteed allocations and purchase on-demand to cover peak times.

The first choice is inefficient. For some users it is even impossible; for example, schools cannot invest on infrastructure large enough to cover rare high needs. The 2nd choice means that providers either set prices (high) to guarantee allocations at demand peaks or renounce to meeting agreed contracts. This is bad for both; consumers may loose revenue and trust in providers or providers may run to losses. The 3rd and the 4th choices are reasonable. They have a fixed part and a variable part subject to market pricing; consumers with high revenue-generating capability will pay the price needed and the providers increase their revenue by

sharing a part of this added value. We foresee a future with mediated cloud market places where consumers opt for the 3rd choice. If consumers do not have a consumption pattern where a stable quantity of resource is required almost all the time they will buy resources when they need them or themselves sell their own resources when capacity exceeds internal demands.

Resources owners may be residential internet users, small organisations and enterprises who do not use all their resources all the time and lease their spare capacity if suitably remunerated. Virtualization technologies and seamless deployment and life-cycle management accompany this future.

The main functionalities provided by the Grid4All market place are:

- **A configurable auction server with two auction mechanisms:**

  Auctions represent a set of rules that determine how exchanges are conducted at a market place. They implement two main functions; (a) compute efficient allocations and (b) determine payments from (to) buyers (sellers). Open market places should support different mechanisms; e.g. a simple yet efficient auction to trade similar VMs for elastic applications; combinatorial auctions to ensure that workflows acquire all required resources at needed relative times; iterative mechanisms allow the auction to elicit preferences on quantity and quality.

  A single auction (or market place) may not scale to trade all types of resources for all consumers and providers. This requires that auction servers be easily deployed and parameterised; e.g. type of resources, the time horizon, the duration and size of the auction itself. The flexible component-based auction server is designed to be deployed by the deployment service described in Chapter 4 and in deliverable D2.2. It facilitates rapid prototyping of new mechanisms and specialization of existing mechanisms. This has been validated during the implementation of two auction mechanisms designed to trade time-differentiated compute resources.

  Basically the auction server accepts requests and offers in the form of structured documents called Bids. Valid bids are retained and cleared. Clearing is the decision process that matches offers and requests to prepare agreements. Finally commodity market clearing prices and payments are computed. We have implemented two mechanisms (i) a combinatorial auction mechanism to trade resource bundles and (ii) an extended double auction to trade multiple units of single type of resources. The Grid4All combinatorial auction mechanism implements an item pricing mechanism. Accurate pricing increases confidence of participants and brings trustable feedback for future operations.

- **A decentralized information service that monitors the market place:**

  The Market Information Service (MIS) acquires economic data from a decentralised market platform and provides it to subscribers. Market feedback is essential for efficient negotiation strategies, to determine starting prices, quantities to request and times to request resources. The implementation combines technologies of distributed information aggregation systems and distributed publish-subscribe models over a structured overlay network and is designed to meet both economic and technical requirements (described in D2.3).

  A key feature is the built-in aggregation and approximation algorithms. The MIS aggregates information (prices, volumes of supply and demand) published by different auctions at different times. Exact aggregation penalizes the performance and is not always essential. The MIS allows users to specify a *Confidence Interval* based on which it automatically increases or decreases the number of sampled data and queries only a subset of nodes (storing publications) while maintaining the desired level of accuracy.

- **Registry and discovery service:**

  This service allows consumers and providers to advertise leasing of resources and querying such leased resources. This SIS has been presented in detail in Chapter 5.

- **Decentralized currency management service:**

  Currency (or money) is the unit of capacity facilitating the interactions and resource exchanges among participants. The Currency Management Service (CMS) developed in Grid4All is a decentralized service that manages payments between participants. Technically it implements a transactional DHT that

ensures atomic account modifications. It serves as a regulatory mechanism and imposes policies on the accounts to ensure that gained currency does not accumulate. The CMS system though implemented and evaluated is not integrated with the other GRIMP components due to lack of resources.

- **Agreement management:**

  The market place decides allocations; i.e. pairs of winning requests and offers. Matched requests and offers represent an Agreement (between provider and consumer) designating the consumer, the provider, the resource description (quantity and quality), leasing times and final payments. An Agreement object contains the end point addresses of winners in the form of URL (unique resource locators). In our implementation, this is required by the leased compute nodes to enact the bootstrap protocol that permits them to join the consumer's VO (overlay).

  In our original design, agreements were expected to be managed by a component called Agreement Manager that ensures that consumers execute payments and providers meet contracts to supply resources. The software prototype does not include the Agreement Manager.

Markets for computational resources differ from that of normal commodities since the former are leased. Time is divided in discrete time slots and resources are indivisibly allocated to a single consumer at a given time slot; with virtual machine technologies, a physical machine may be logically 'divided' into multiple isolated containers, each leased independently.

Consumers (providers) send orders (bids/offers) that include the following attributes:

- Start time: the time when resource is required (offered).
- End time: the latest availability of resource or the latest time of allocation.
- Duration: valid only for consumers and gives the duration for which the resources are needed.
- Resource type and attributes of the specified resource(s): Providers submit exact values where as consumers request resources that satisfy these attributes (for example minimum CPU speed).
- Price: For a provider this gives the minimum unit price per unit time and for a consumer it gives the maximum price for the requested bundle of resources.

Bids may express requests such as "Consumer C1 bids for 4 machines (physical or virtual) of type T1 for 6 hours where the earliest time of availability should be 12:00 and the latest should be 22:00 and is willing pay at most 5 Euros". They may be composed using the conjunction, disjunction or the exclusive disjunction operators. Such compositions allow consumers to express workflows, e.g. "Consumer C1 bids for 8 machines (physical or virtual) of type T1 for 2 hours between 12:00 and 16:00 and for 4 machines of type T2 for 3 hours each between 15:00 and 19:00 for 4 Euros, or otherwise 6 machines of type T3 for 1 hour each between 12:00 and 14:00 for 5 Euros".

6.3 Architecture

The main prescription of the Grid4All platform is to use Fractal model to design applications. This favours a uniform deployment and management model where by the tools described in chapter 4 can be used. The design tenets that the GRIMP components adhere are: (a) extensibility (b) ease of deployment and (c) mechanisms pertinent for democratic grids i.e. simple to use and easy to interpret.

6.3.1 Configurable auction server (CAS) and mechanisms

The server is built as a set of components interacting through well defined interfaces and described using Fractal ADL (Figure 6). Two control patterns are supported, iterative auctions where bids are replayed by participants until termination conditions are reached and single-shot auctions. CAS can execute either in a continuous mode (it perpetually waits for and clears bids) or in a scheduled mode (clearing occurs at prescribed events).

Auctions are started on demand (see Figure 7 that gives a portal interface). The instantiated auction is accessed through a small set of programming interfaces of the following categories:

- Configuration interfaces to configure the auction instance; specification of times regulating the auction, resource types, the acceptable bundle configurations and the lease time horizons.
- Registration interfaces that must be used by the participants to authenticate.
- Functional interfaces to submit bids and set up agreement notification handlers.

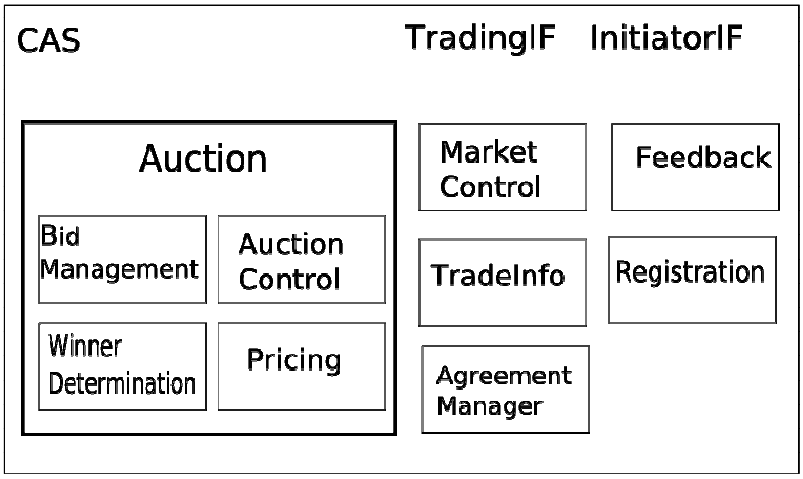


Figure 6 Configurable auction server

Most CAS components are generic. The auction mechanism is itself embedded in the Bid Management, Winner Determination (computes allocations) and Pricing component (computes prices).

Auction deployment and market factory

The auction server orchestrates the bidding process; matches offers and bids, computes legible prices and generates Agreements. The market place should include a *Factory* that provides services to instantiate auctions. This module called the *Market Factory* is not implemented due to lack of resources.

Using remotely allocated (leased) resources

Current models of using Grid and Cloud resources support the following resource usage models:

- Grids: consumers, in fact members of recognized VOs submit jobs to Grids described in languages such as JSDL (Job Submission Description Language). Executables and input (output) files are transferred between Grid and client-specified nodes. Jobs wait in queues before being executed; RMS[10] accepting advanced reservations guarantee that submitted jobs execute at the agreed time.
- Clouds: clients lease virtual machines (VM) hosted on physical machines located at the provider site. Minimum lease duration is typically fixed (1 hour). Allocated virtual machines may be configured to have both private and one or more public IP addresses. The latter is required to deploy the application (or service) on the allocated VMs and interface with it. In case of infrastructure clouds (IaaS), the application deployment and life-cycle management is managed by the clients.

Our model to use leased resources is different. Leased nodes (computers or VMs) are expected to *join* the VO overlay of the buying consumer. Joining is a two-way protocol between a joining node and a host node (admitting peer). We assume that Grid4All containers (see 4.4) are installed and deployed (in a passive

[10] Resource Management System

mode) on leased nodes and are configured to join the overlay via a bootstrap node designated by the URL given in the Agreement. Other protocols may be used, e.g. SSH, or any recent peer-to-peer join protocols.

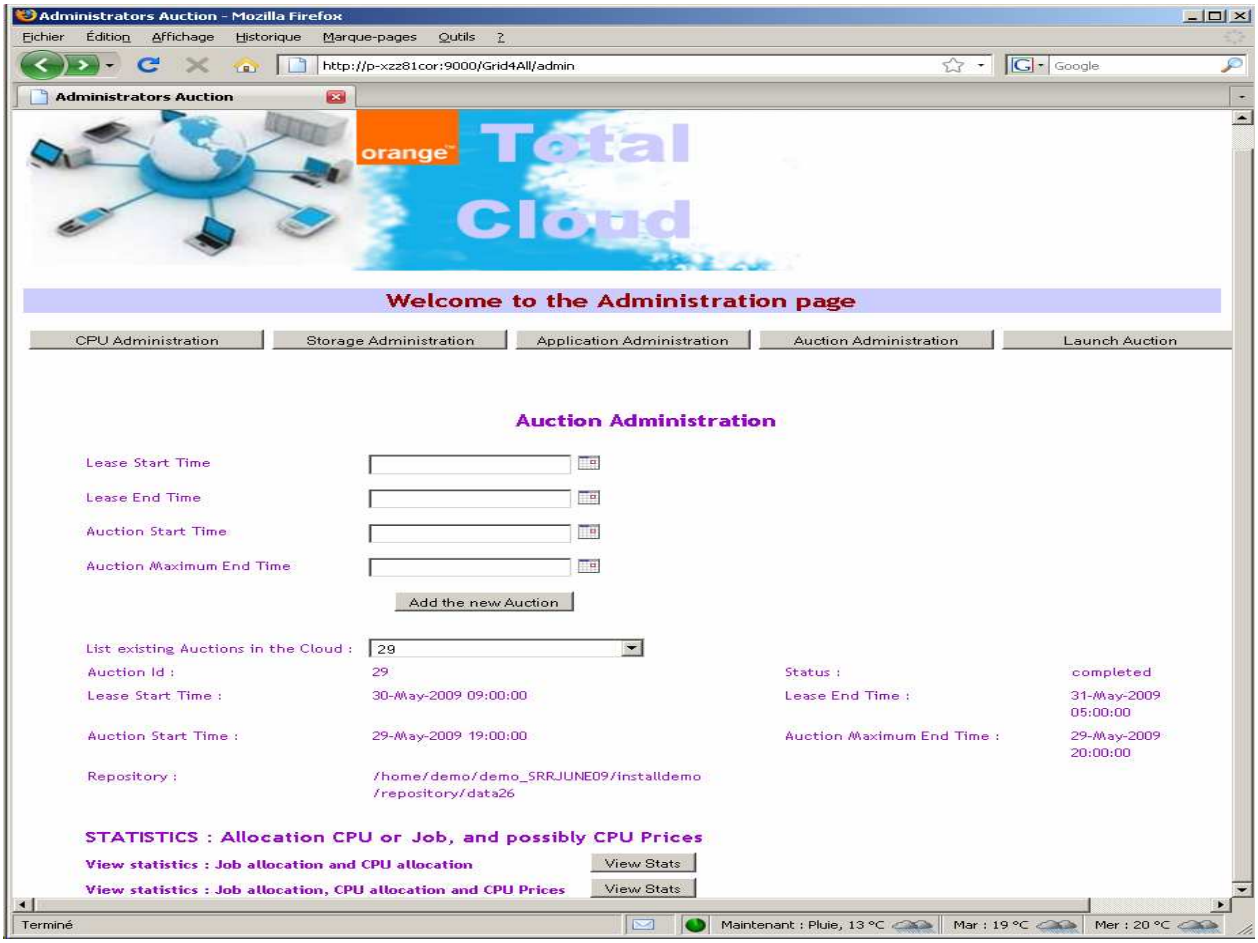


Figure 7 Auction configuration

6.3.2 Market Information Service (MIS)

The technical challenge for a decentralized market information system is to meet the economical requirements in combination with technical requirements of distributed systems. The economical side requires disclosure of aggregated and individual data within acceptable delays. The technical realization has to cope with resource churn and to scale in regard to traders and traded products. Traders subscribe to information such as price of traded products.

The MIS consists of four layers; the Market Information System (MIS) application, the Distributed Market Information System (DMIS), Routing, and DHT, as described in the Figure 8. Each layer copes with different technical or economical requirements. The MIS layer provides interface between publishers and subscribers to the system. The Routing layer provides the routing functionalities used by the DMIS. The structured overlay layer enhances the scalability and robustness of the system. It operates as the communication layer and is used by the DMIS. The DMIS itself implements the core functionality, i.e. maintain subscriptions and publications and query processing.

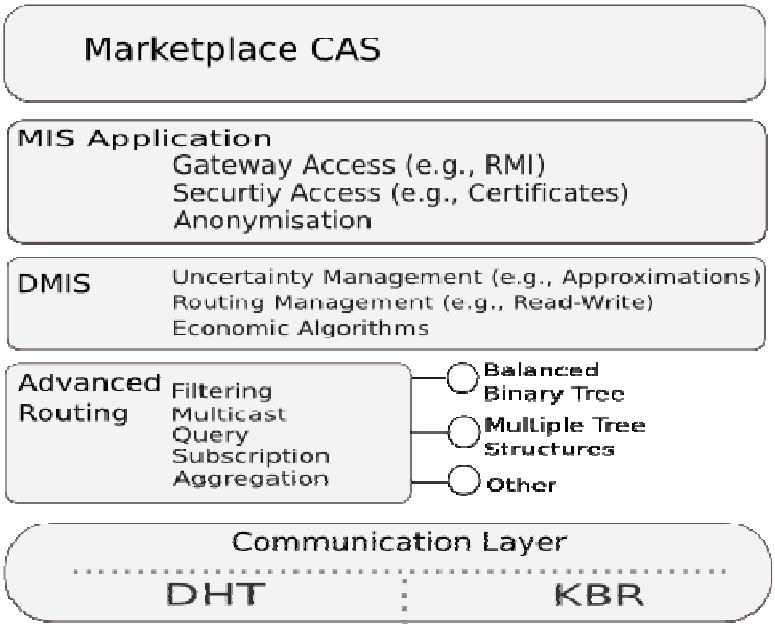


Figure 8 MIS Architecture

The core functionalities of the system are processed in the **Advanced Routing layer**:

- Filtering: Messages are forwarded to interested nodes. For example an event with a price of 4 is only sent to nodes interested in events with a price higher than 3[11].
- Multicast: Messages are sent to subgroups of nodes interested in a given topic.
- Query: This function executes a query for a read-dominated value within the market place. It follows either an epidemic structure, binary-tree structure or multi-tree structure. The current prototype implements a query in a binary-tree in an ordered subset of clients. An objective is to implement multiple-trees, which have a higher robustness but also increase the amount of messages.
- Subscription: This is the process to join a certain topic or content, and accordingly to obtain interested information.
- Aggregation: Aggregation decreases the number of messages. Some are simple (maximum, minimum), and others more sophisticated e.g. average prices. Complex queries with a combination of parameters are also supported (select price where storage > 100 GB and memory > 3 GHz).

The **DMIS layer** provides and coordinates core functionalities for the trader (client). The trader can invoke the API methods to subscribe, query or publish information. It provides handlers like the SubscriptionHandler, QueryHandler or RequestHandler for messages returned to the trader. Non-functional methods are provided to bootstrap to the network.

The **MIS layer** provides functions necessary to integrate with different platforms, through one or more gateway nodes that are connected to the DMIS; e.g. connecting to the DMIS via Web Services technology.

6.4 Implementation

The MIS is provided as a Java library that embeds the communication protocols and access interfaces. It is encapsulated in a Fractal application and its architecture is described using the Fractal ADL to unify the deployment process within the Grid4All environment. The services of the MIS are accessed by the CAS to publish the market data (e.g. price by type of resource at specific times, supply and demand volumes) using

[11] This functionality is not implemented in the current version.

RMI. Finally a direct access to the MIS Java library enables an easier deployment for functional testing and simulations with a subset of other components such as CAS.

The CAS is provided as a Java library. Now it does not require major non-Grid4All software since it is not integrated with subsystems such as payment, identity and authentication management. CAS architecture is described using an ADL and deployed by tools described in chapter 4. Earlier version of the combinatorial mechanism needed the CPLEX server, but the final version implements efficient heuristics-based algorithms. Decision to use the CPLEX servers could take into account the time allowed to clear the auction, the availability of licences, the leasing times (e.g. if sufficiently far away then the CPLEX server could be used).

6.4.1 Technical considerations

Grid4All combinatorial auction mechanism

This type of auction supports complex combinatorial requests for resources needed to execute workflows or rigid applications that require bundles of different resources in precise (at least minimum) quantities and durations. Exposure problems where jobs are left hanging with a subset of needed resources are avoided by issuing combinatorial requests for all needed resources. Integrating the time dimension in requests assimilates this class to job scheduling. The disadvantages of combinatory auctions are (a) lack of transparency and (b) resolution complexity (allocation and pricing). In this section we use the term *Job* to refer to a request submitted by consumers and *Bundle* to refer to the offer submitted by providers. A job is executed over a time-horizon (earliest and latest times) and specifies the total duration for which resources are required to execute the application within this horizon. A job consists of *parts*, each part may execute in its own time horizon. This allows many complex workflows to be represented. A Bid of a consumer is an exclusive disjunction of jobs. A Bid of a provider is a set of disjunctive bundles, i.e. one or more of the elements of the set may be matched to Jobs.

The allocation model had been described in detail in D2.3. It maximises social welfare, subject to two secondary objectives (i) maximize number of allocated jobs (ii) allocate bundles as early in time as possible. Here we present some heuristic algorithms and the pricing model.

The first heuristic resolves the continuous relaxation of the original integer program and rounds the obtained solution. If a job is allocated, i.e. all its tasks are allocated then it is marked as allocated or otherwise rejected (if the solution has a value inferior to 1). A second heuristic greedily sweeps the set of jobs. For each job the set of bundles are swept and the first possible allocation is marked. The way to order jobs, bundles and the time slots is relevant. Different methods have been considered to determine the priority of a job, e.g. higher the priority (a) higher the surplus that it is expected to produce (b) higher the difficulty to match the job. Consider the job *nj*, (j[th] job of consumer n), the values by which jobs may be ordered are:

- ✓ Valuation, i.e. *Val* (*nj*):=v^{nj}.
- ✓ Valuation per unit of job (ValUnit) determined as:

$$V_u = \frac{v^{nj}}{\sqrt[|R|]{\prod_{r \in R} A_r^{nj}}}.$$

Where A_r^{nj} is gives the total quantity of resources used by the job and is given as:

$$A_r^{nj} = \sum_k \in K^{nj} a_r^{njk}.q^{njk}.$$

A variation is to use the square of the valuation of the job (ValUnit2) to accentuate the differences between jobs.

- ✓ The size of the job, i.e.

$$\mathrm{Size}(nj) := \sqrt[|R|]{\prod_{r \in R} (\sum_{k \in K^{nj}} a_r^{njk}.q^{njk})}.$$

- ✓ Flexibility, i.e. the ratio of average duration of the job over the amplitude of the interval (time horizon)

$$\mathrm{Flex}(nj) := \frac{\sum_{k \in K^{nj}} q^{njk}/|K^{nj}|}{l^{nj} - e^{nj} + 1}.$$

- ✓ Average number of possible allocations for the tasks of the job. Let Alloc(*njk*) be the set of compatible allocations (if all constraints of all tasks of a job are satisfied by a subset of bundles) then the average number is determined as

$$\mathrm{NbAlloc}(nj) := \sum_{k \in K^{nj}} |\mathrm{Alloc}(njk)|/|K^{nj}|.$$

- ✓ The expected cost of a job determined as

$$\mathrm{ExpCost}(nj) := \sum_{k \in K^{nj}} \sum_{mbt \in Alloc(njk)} (\sum_{r \in R} a_{r,0}^{njk} . p_r^{mb} . q^{njk})/\mathrm{NbAlloc}.$$

- ✓ The expected surplus, i.e. difference between valuation and the cost determined as

$$\mathrm{ExpSurplus}(nj) := \mathrm{Val}(nj) - \mathrm{ExpCost}(nj).$$

Similar methods are proposed to order the bundles and time slots. Time slots are ordered in increasing order of load (load is the ratio of aggregated resources required at a specific time over that aggregated resources offered at that time). Ordering of bundles may be determined by:

- Increasing order of reservation price $\mathrm{Res}^k(b) = \sum_r (a_r^k . q^k . p_r)$
- Average reservation price ResAv (b)= $sqrt_r (\prod_r p_r)$
- Random
- ResQualQuant, i.e. start in increasing order of average reservation price (ResAv), then average quality and finally by average quantity.

Once an order is fixed, greedy and gradient descents algorithms are used to match jobs (and their tasks) to the bundles. As an example Table 5 gives evaluation results with different job and bundle ordering methods. A_100_12, B_100_12, C_100_12 are three generated instance sets (jobs and bundles). Each corresponds to 100 bundles and 100 jobs specified over a 12 hour time horizon. %UB gives the upper bound on the welfare as compared to an exact solution. The time taken for each resolution is in the order of minutes (~6 minutes) as compared to that of the exact solution (~3 hours), making these heuristics good candidates.

Ordering	**VAL + Random**			**Val + Res**			**Val + ResQualQuant P=20**			**Val + ResQualQuant P=∞**		
Instance	welfare	#jwin	% UB	welfare	#jwin	% UB	welfare	#jwin	% UB	welfare	#jwin	% UB
A_100_12	31084	80	79,22%	32785	82	83,55%	33156	79	84,50%	32983	78	84,06%
B_100_12	34447	84	81,58%	35750	82	84,66%	36001	82	85,26%	35875	82	84,96%
C_100_12	29260	73	78,69%	31778	78	85,46%	31939	74	85,89%	31917	76	85,83%
AVG	**31597,00**	**79,00**	**79,83%**	**33437,67**	**80,67**	**84,56%**	**33698,67**	**78,33**	**85,22%**	**33591,67**	**78,67**	**84,95%**
Ordering	**ValUnit2 + alea**			**ValUnit2 + Res**			**ValUnit2+ ResQualQuant P=20**			**ValUnit2+ ResQualQuant P=∞**		
Instance	welfare	#jwin	% UB	welfare	#jwin	% UB	welfare	#jwin	% UB	welfare	#jwin	% UB
A_100_12	31357	81	79,92%	32554	80	82,97%	32859	80	83,74%	32909	82	83,87%
B_100_12	35185	82	83,32%	36132	83	85,57%	36101	82	85,49%	36216	81	85,77%

C_100_12	29574	75	79,53%	31940	79	85,89%	32217	80	86,64%	32252	78	86,73%
AVG	**32038,67**	**79,33**	**80,92%**	**33542,00**	**80,67**	**84,81%**	**33725,67**	**80,67**	**85,29%**	**33792,33**	**80,33**	**85,46%**

Table 5 Comparison of different heuristics to resolve CA

Three algorithms to match jobs to bundles have been implemented and evaluated; (i) repeated greedy (GREEDY) (ii) gradient descent (DESCENT) and (iii) combination of these two (GRASP). The GREEDY algorithm starts from an initial ordering, e.g. ValUnit2 and then permutes two jobs drawn randomly and exchanges their position. The selection is slightly perturbed by drawing random numbers according to a distribution defined by values of ValUnit2. The DESCENT algorithm starts with a solution set and moves towards the closest improving neighbour. Over a sizeable execution run, this method has shown to improve the number of winning jobs without displacing the optimal jobs (winning jobs remain winners). GRASP combines the two approaches by doing a gradient descent from the best solution achieved by the GREEDY algorithm.

Conclusively the quality of allocation obtained using the proposed heuristics makes it a promising candidate for a market place as designed in Grid4All.

Pricing is the process by which the auction determines the payment received from (paid to) winners. Good pricing methods help accrue participant confidence in the market place and help them understand the market status. Bundle or per-resource prices may be computed. The latter improves clarity since participants can calculate the price of new bundle configurations by linear summation. Another dimension is time; prices as a function of time render this information more legible. Consumers may adjust their requests to times that best improve their utility. We determine prices of resources as a function of time (in a discrete time horizon) such that they approach the market clearing pricing, i.e.:

- ✓ The payment done (or received) is equal to the sum of prices of resources in the job (bundle).
- ✓ A losing job has a valuation that is inferior to the sum of price of resources required.
- ✓ A losing bundle has a reservation price higher than the sum of prices of resources that it offers.

The proposed model preserves budget-balance, i.e. sum of payments to suppliers and that received from consumers is non-negative and individual rationality, i.e. no participant pays more (or paid less) than he can afford. In this method, we find the price π_r of each resource *r* such that winning consumers will pay a linear sum. If *n* represents consumer and *j* represents the job of the consumer, then this consumer's payment is:

$$\nu^{nj} = \sum_{r \in nj} \Pi_r$$

Similarly a seller *m* will receive for his bundle *b*:

$$\mu^{mb} = \sum_{r \in mb} \Pi_r$$

The individual rationality property is described as:

$$\forall n, j \text{ gagnants } \nu^{nj} = \sum_{r \in nj} \Pi_r \leq v^{nj}$$

$$\forall m, b \text{ gagnants } \mu^{mb} = \sum_{r \in mb} \Pi_r \geq r^{mb}.$$

This is some what ideal since the same type of resource may be present (traded) in different winning bundles. Fixing the price of a resource type may imply that the above inequalities are not verifiable, i.e. there may not be a solution such that the IR property is verified for all bundles.

Example: 4 buyers, 4 sellers, 4 resources a, b, c and d.

n_1	ab	10	m_1	ab	8
n_2	ac	6	m_2	ac	4
n_3	b	6	m_3	b	5
n_4	c	6	m_4	c	5

The optimal allocation couples each n_i with m_i and has value 6. The equations for the buyers give and sellers are:

$$\Pi_a + \Pi_b \leq 10 \ ; \ \Pi_a + \Pi_c \leq 6 \ ; \ \pi_b \leq 6 \text{ et } \Pi_c \leq 6.$$
$$\Pi_a + \Pi_b \geq 8 \ ; \ \Pi_a + \Pi_c \geq 4 \ ; \ \pi_b \geq 5 \text{ et } \Pi_c \geq 5.$$

This results in the following inconsistencies:

$$\Pi_a \geq 8 - \pi_b \geq 8 - 6 = 2$$
$$\Pi_a \leq 6 - \pi_c \leq 6 - 5 = 1.$$

Hence a simple approach will not give a correct solution. We relax the closeness of the payments to market clearing prices by two families of variables epsilon that controls this slack.

$$\mu^{mb} \geq \sum_{r \in mb} \Pi_r - \epsilon_1^{mb}$$
$$\nu^{nj} \leq \sum_{r \in nj} \Pi_r + \epsilon_2^{nj}.$$

$$\mu^{mb} \leq \sum_{r \in mb} \Pi_r + \epsilon_3^{mb}$$
$$\nu^{nj} \geq \sum_{r \in nj} \Pi_r - \epsilon_4^{nj}.$$

Consider M_W and N_W the set of winning consumers and providers. For every $m \in M_W$, $B^m{}_W$ is the subset of B_m of winning bundles. $J^n{}_W$ gives the subset of J_n of winning jobs. Similarly M_L and N_L are losers ($M_W \cup M_L = M$, $N_W \cup N_L = N$) where $B^m{}_L$ and $J^n{}_L$ denote losing bundles and jobs: $B^m{}_L = B^m B^m{}_W$, $J^n{}_L = J^n$ if $n \in N_L$, else $J^n{}_L = 0$. Due to the XOR operator, a job is a loser only if none of the other jobs of the same consumer wins. We assume that the clearing is done and allocations are optimal. $\delta^{mb}{}_r$ denotes the quantity of resource r of bundle mb that is consumed in the optimal allocation. μ^{mb} denotes the payment to winning bundle b. $\nu^{nj} \in R^+$ denotes the payment by winning job j. $\Pi^{r\alpha} \in R^+$, with $r \in R$ and $\alpha \in A_r$ denotes the price of resource r. The pricing is then formalized as:

$$\min_{\pi, \mu, \nu, \epsilon, Z} Z \text{ subject to the constraints:}$$

$$\sum_{m \in M_W} \sum_{b \in B_W^m} \mu^{mb} = \sum_{n \in N_W} \sum_{j \in J_W^n} \nu^{nj}$$

$$\sum_{m \in M_W} \sum_{b \in B_W^m} \mu^{mb} = \sum_{n \in N_W} \sum_{j \in J_W^n} \nu^{nj}$$

$$\forall m \in M_W, \forall b \in B_W^m, \mu^{mb} \geq \sum_{r \in R} \sum_{t} \delta_{r,t}^{mb} \Pi_{r,t} - \epsilon_1^{mb}$$

$$\forall n \in N_W, \forall j \in J_W^n, \nu^{nj} \leq \sum_{k \in K^{nj}} \sum_{r \in R} a_r^{njk} \sum_{t=t_0^{njk}}^{t_0^{njk} + q^{njk} - 1} \Pi_{r,t} + \epsilon_2^{nj}$$

$$\forall m \in M, \forall b \in B_L^m,$$

$$\sum_{r \in R} d_r^{mb} \sum_{t=c^{mb}}^{f^{mb}} \Pi_{r,t} - \epsilon_3^{mb} \leq (f^{mb} - c^{mb} + 1) \sum_{r \in R} d_r^{mb} p_r^{mb}$$

$$\forall n \in N_L, \forall j \in J^n,$$
$$\sum_{k \in K^{nj}} (\sum_{r \in R} a_r^{nj} \sum_{t=e^{nj}}^{l^{nj}} \Pi_{r,t}).q^{njk} / (l^{nj} - e^{nj} + 1) + \epsilon_4^{nj} \geq v^{nj}$$
$$\forall m \in M_W, \forall b \in B_W^m, 0 \leq \epsilon_1^{mb} \leq Z$$
$$\forall n \in N_W, \forall j \in J_W^n, 0 \leq \epsilon_2^{nj} \leq Z$$
$$\forall m \in M, \forall b \in B_L^m, 0 \leq \epsilon_3^{mb} \leq Z$$
$$\forall n \in N_L, \forall j \in J^n, 0 \leq \epsilon_4^{nj} \leq Z$$
$$\forall m \in M_W, \forall b \in B_W^m, \mu^{mb} \geq \sum_{r \in R} p_r^{mb} \sum_t \delta_{r,t}^{mb}$$
$$\forall n \in N_W, \forall j \in J_W^n, \nu^{nj} \leq v^{nj}$$
$$\forall r \in R, \Pi^{r,t} \geq 0$$

The above model defines payments and prices (μ, ν, Π) such that Z is minimized and sets upper bounds on distortions (epsilon). It balances budget and ensures that the market clearing property is not violated. It has been designed after studying some important pricing schemes. The classical Vickrey-Clarkes-Groves (VCG) pricing gives each participant a discount in proportion to his contribution to the welfare, but this method is complex (the WDP should be recomputed as many times as number of participants) and budget-imbalanced. K-pricing is a scheme by which a fixed parameter 'k' determines the distribution of surplus between the consumers and the providers. A provider receives surplus proportional to the bundle that it sells; but it is not clear how to determine the contribution of each bundle. A recent design in combinatorial auctions proposes market clearing prices or close approximations. This method finds prices such that the difference between the values of losing bundles and the linear sum calculated from derived prices is minimized. We have extended this to apply it to auctions where multiple time-differentiated units of multiple resources may be traded. The Table 6 provides a comparative evaluation of these methods.

Pricing	type	IR[12]	BB[13]	IC[14]	T[15]
VCG	BP	Yes	No	Yes	No
k-pricing	BP	Yes	Yes	No	Yes
BB VCG	BP	Yes	Yes	No	No
1-side VCG	BP	Yes	Yes	for buyers	No*
Approx MC	IP	Yes	Yes	No	Yes

Table 6 Comparative evaluation of pricing policies

Extensions to the classical double auction

The classical double-auction mechanism is employed in markets that trade in single type of resources such as for example processing or storage capacity or more generally a virtual machine (of a pre-agreed

12 Individual Rationality
13 Budget balance
14 Incentive Compatibility
15 Tractability

configuration). Simple heap algorithm is used to implement the matching; the highest bid (request) is matched to the lowest ask (offer). However when trading time-differentiated resources, an auction is instantiated for every time slot. This is not efficient. Ideally users would like to submit requests (bids) for a time horizon within which they desire resources to be allocated.

- A typical bid (request from consumer) should be expressible as: "4 virtual machines for 3 hours each between 10:00 and 18:00".

A bid may be matched by one or more ask that respects the technical (resource characteristic and time horizon) and economic constraints (price).

We have designed and implemented a novel algorithm called the MLDA (multi lane double auction) that maintains this simplicity of expression to users. It computes an efficient allocation, i.e., finds matching requests and offers that maximize social welfare and the number of allocations. This algorithm works by organizing received bids and asks in multiple lanes, each lane corresponds to a distinct time slot. Asks and Bids are classified as winners and losers according to their prices. Bids may be precise, i.e. specify the exact times at which resources are required bids that need resources or imprecise, i.e. allow a time zone within which resources may be allocated. Asks are necessarily precise; a provider explicitly offers resources for a defined time horizon.

Precise bids are routed to the corresponding lane. Imprecise bids are inserted in lanes that maximize the social welfare. If multiple lanes are candidates (due to impreciseness of the bid), the lane that does not displace a previous winner (moving it to become a loser) is selected. The objective of this design that coordinates the matching in different time slots is to ensure that (a) priority inversion does not happen, i.e. a lower bid (higher ask) does not displace a higher bid (or lower ask) (b) an imprecise bid does not take the limited places (time slots) allowed to a precise bid.

Market information service and routing algorithms

The MIS architecture has been designed to meet both the economic information requirements and that of scalability and robustness of a large-scale distributed environment. The implementation employs tree-based routing structures and approximation algorithms to improve performance. Measurements on the proof-of-concept implementation in a large-scale deployment have confirmed the impact of approximation.

Aggregation is done using tree-based routing mechanisms. The topic-based MIS aggregates nodes holding events of a desired topic as a B-Tree. It improves efficiency of the approximation mechanism since only a sub-set of nodes are queried. The uncertainty in the quality of returned information is automatically managed based on a Confidence Interval specified by the subscriber. The system automatically adjusts (increase or decreases) the number of messages to get the sampled data. Complex queries are partially supported; joins across different topics are not implemented.

The Figure 9 shows how information published by auctions may be of use; two sets of results (a) the allocation statistics, i.e. number of jobs submitted and allocated at each hour (b) prices of different types of CPUs at each hour of a 24 hour interval. It helps participants determine peak periods, i.e. time slots corresponding to peak prices. Participants, both consumers and providers can plan their capacity based on analysis of historical prices.

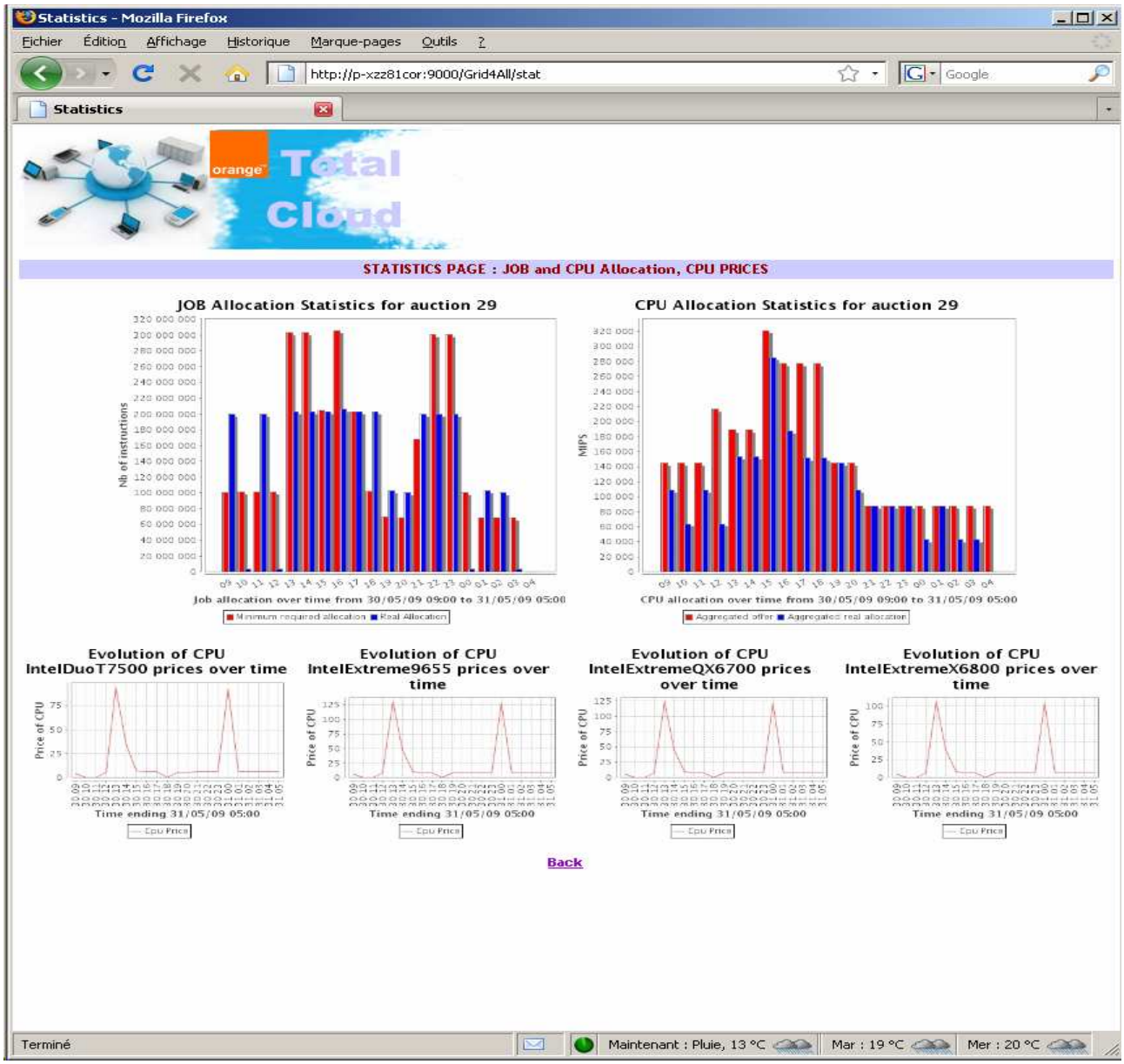


Figure 9 Allocation and price statistics

6.5 Configuration

The GRIMP tools and services can be deployed using the Grid4All deployment services described in section 4. The distributed MIS organised as a peer-to-peer overlay, mainly needs the port number and the IP address of the bootstrap node (considered as a reference to the peer-to-peer network itself). This should be configured on each of the MIS node. MIS is packaged and deployed using the Jade and Fractal deployment service and offers its services through an RMI interface for auction servers that execute on the same node.

6.6 Used technologies

The MIS implementation uses the FreePastry (http://freepastry.org/) library. This library contains a P2P Key-Based Routing (KBR) mechanism and a P2P-based publish-subscribe mechanism. We use and modify these techniques provided by Pastry for the communication layer to obtain a large-scalability of our system. Using this existing library enables more possibilities in comparing with other state of art implementations in form of simulations and prototypes.

The auction server and its mechanisms are Fractal-based applications and require Jade-based deployment service. The combinatorial auction has been implemented in two flavours (a) using CPLEX server and (b) using heuristics and meta-heuristics based algorithms. Rigorous evaluations over a wide range of data sets show that heuristics achieve almost 85% of efficiency for most input data sets and in a much shorter time. Auction server can decide the implementation it uses based on parameters such as: time to execute, availability of CPLEX licence etc.

6.7 Other related G4A components

- **Information service and registry:**

  Market place participants use the G4A-SIS services (chapter 5) to discover appropriate auctions matching resource characteristics (of resources traded at that auction) and economic properties (prices and other auction properties).

- **Reservation management and remote execution service:**

  Application managers executing on Grid4All VOs can use the reservation management API to lease resources. The Reservation Management sub-system (chapter 4), manages the interactions with the market place and notifies application managers when leased resources join the virtual organisation.

- **Deployment:**

  With the Reservation management the deployment service (chapter 4) completes the steps to provision resources; managers allocate resources and deploy (install, configure, activate) application components.

- **Niche Distributed component management system:**

  Niche (reported in D1.5 and D1.6) can be used to develop self-managing applications. Applications can be monitored and management components can detect load spikes and actuate provisioning mechanisms: allocate new resources, deploy needed components and reconfigure the application architecture to match current load.

6.8 Current status

This section has outlined the core functionalities of the Grid4All market place that are implemented. To deploy realistic Grid market places additional subsystems are needed. The Figure 10 below gives an idealized market place. The project has focussed on the first three. Grey lines indicate mandatory functionalities. Orange entries are features that provide additional value.

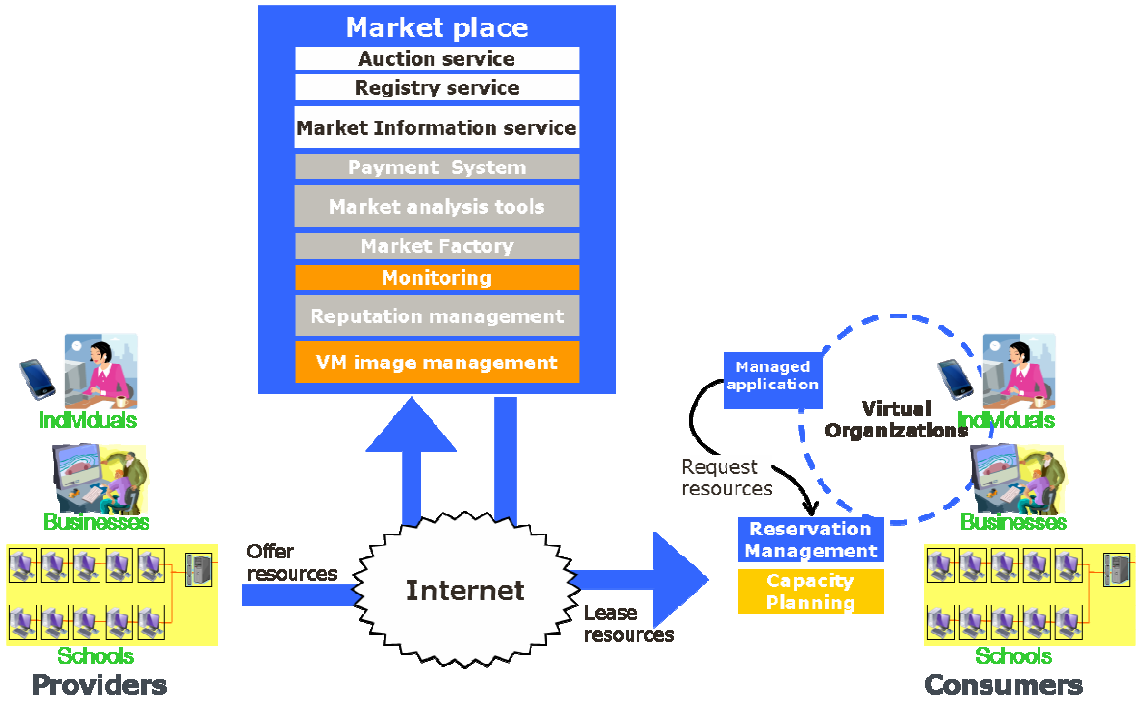


Figure 10 Ideal resource market place

6.8.1 CAS and auction mechanisms

The prototype implementation of CAS ensures the basic functionality of an auction server, i.e., configuration, registration, acceptance and validation of bids (and offers), allocation decisions, pricing and feedback. Moreover the architecture has permitted rapid prototyping of different auction mechanisms and algorithms without requiring complete modification of code.

Nevertheless before becoming a system that can be deployed more components are needed:

- ⇒ Market factory that offers services to deploy auctions on demand allowing users to specify the characteristics (economic) of the desired auction mechanism,
- ⇒ Robust and flexible agreement management and tracking of agreements generated by the auction,
- ⇒ Authentication and authorization of participants.

Concerning the designed mechanisms, the MLDA algorithm has been evaluated to compare economic and computational efficiency as compared to independent multiple instances of a single double auction. The comparison done is between (a) M instances of the classical double auction where each instance auctions computational capacity for a specific time slot of the entire time horizon and (b) one instance of the MLDA that auctions computational capacity for the entire time horizon. In case of (a) precise and imprecise bids are randomly distributed to one instance and in case of (b) to the single deployed MLDA. The MLDA shows significant improvements on economic efficiency (aggregated social welfare), number of allocations and also overall computational time measured in seconds. The drawback was in terms of memory that is required to maintain its data structures.

6.8.2 MIS

The prototype implementation of MIS ensures the basic functionality described in chapter 6. The interfaces are defined and validated with the other components that use these interfaces, namely the configurable auction server (CAS). Mock testing has been done to show correct functioning. A stand-alone integration with CAS has been tested and evaluated.

The MIS has been evaluated in regard to technical aspects such as scalability and quality of approximations. Approximation methods promise a significant improvement of the number of messages and network load in regard to related work.

The main role of the MIS is to provide participants aggregated and summarized economic information from the market place, such as prices, volume of supply and demand. Participants, i.e. consumers and providers may then use this information to trace their strategies. The main objective of consumers is to acquire resources at quantities and qualities at appropriate times such that their utility is maximized. Information from the market may help consumers to adjust their demands in these dimensions. The objective of providers is to ensure that they get maximal remuneration for their perishable resources (perishable in that if the CPU is not consumed at a time then it is lost). Providers have their own local requirements, i.e. internal needs for the resources that they own. Knowledge of market dynamics permit them to plan their capacity, i.e. when to lease and at what prices. This will be focus of future work.

6.9 State of art comparable results

The pertinence of a resource market place should be analyzed in the context of Clouds and we discuss this in section 6.12. This section gives an overview of some outstanding projects (there are no products or operated platforms today). Comparison is not trivial since each project (a) has non overlapping requirements (b) provide different technologies, (c) advocate different architectures and (d) are not deployable systems.

Tycoon

Tycoon (http://tycoon.hpl.hp.com/) is a market-based system for managing compute resources in distributed clusters like PlanetLab, the Grid, or a Utility Data Center (UDC). Users have a limited credit supply. Consumers pay providers to use computer resource. Providers resources can, in turn, buy resources with their earnings. Tycoon developers show that their mechanism (deciding allocations) is more efficient than standard priority or proportional share systems. It is currently a prototype implemented using Linux and Xen Virtual Machine technology.

Tycoon is a closed economy, i.e. users earn currency by selling resources that they can later use to buy resources. This is not practical when different types of resources may be exchanged. Tycoon mechanism is applicable only for elastic applications where virtual machines can be added incrementally.

Sorma

Sorma (www.ist-sorma.eu) is an FP6 project whose objective is to research and develop a complete ICT market place. The project has developed a number of components (SLA monitoring, payment, policy-based bid generation) that may be adapted to Grid4All as well. Sorma architecture requires adaptations to Grid RMS to make them economy aware, i.e. understand Sorma modifications to JSDL and WS-Agreement protocols to include economic properties. These approaches are somewhat outdated in the context of today's Cloud computing where resource allocation models differ from traditional queue-based Grid RMSs.

Sorma has investigated in a rich set of market and pricing mechanisms for different use cases; batch oriented applications, real-time applications. Extensive work has been done on SLA enforcement. They provide decision making tools for participants to specify their resource requirements (bids).

We believe that there is room for collaboration between the two projects to share results.

GridEcon

This FP6 project has focused on building rich tools for market places, in contrast to Grid4All which had a wider set of ambitions for democratic grids. This project believes in a future where compute resources will be traded at a market place (consumer one day will probably be a provider another day). It has designed

decision making tools such as (a) risk analysis (b) insurance management against failing providers (c) capacity planning by consumers to help identify when and what quantities of resources should be acquired at Grid markets.

GridEcon (www.gridecon.eu) mechanisms are suitable to allocate virtual machines for elastic applications, i.e. those adapting to the final allocated quantities. This model is not optimal for workflows deployments. The project has produced a rich set of analysis on (a) conditions necessary and sufficient for grid economies such as technico-economic factors (b) future markets for grid resources and stable price maintenance.

We believe that there is room for collaboration between the two projects to share results.

GRACE

This project has done extensive work on Grid Economy and is conducted by the GRIDS (Grid Computing and Distributed Systems) laboratory of Melbourne University in Australia. It was the first to identify that a major challenge for next generation Grid computing is creation of a competitive Grid Market place that regulates supply and demand and gives the right incentives to participants (suppliers and consumers). It have produced a set of Grid middleware; some of these are adapted to a Grid economy, i.e. market directory for publishing and searching for available services including price related attributes, price aware job scheduling etc. It has experimented monitoring tools to observe market behaviour, e.g., what kind of requests and offers arrive at a given serving node to help participants to adjust their own offers/requests. Market Resource Brokers are services (per class of applications), that aid consumers to discover resources, match jobs to resources, deploy jobs and monitor their progress. The proposed architecture is rigid and centralized though recent announcements indicate that it is evolving towards distributed/decentralized architectures.

6.9.1 Discussions on state of art

In the light of current state of art, we note that many projects have produced interesting results, but on a practical and pragmatic aspect, lack of interoperability and/or extensibility renders these results unusable in a new context. Sharing of results from state of art is restricted towards reuse of published ideas and algorithms. A second point to note is that even though large scale sophisticated and rich service platforms (telecommunication services, e-commerce services), have existed since a long time, there is currently little effort done to leverage at least expertise if not the architecture, platform and tools.

6.10 Software

The software can be obtained from the Grid4All source management site at: https://scm.gforge.inria.fr/svn/grid4all/wp2/GRIMP. Furthermore, the MIS is based on the following project DMIS, information and source code of that project can be found at: https://code.ac.upc.edu/project/dmis.

The source of the combinatorial auction may be obtained on demand.

6.11 Innovative usages

The market place that we envision is an environment that allows small enterprises and organisations and individual domestic users of Internet to trade their resources. This does not preclude large organisations to either offer their spare capacities or to lease capacities during transient spikes. With emergence of clouds (infrastructure, platform and services), new business models will arise. Entities such as telecom operators will act as brokers and offer value-added mediation services. Telecom operators are well-positioned to play this role since they (a) control the network (b) have a large client base (c) possess identity management services (d) possess scalable technologies for metering and billing usages.

The project has delivered a set of tools that can be considered as a starting point towards deployment of a full-fledged market place. The market place itself may be deployed and manages using tools provided by WP1 and WP2. Consider a telecom operator who is able to provide a small but stable set of nodes on which

essential market services are deployed (a) SIS (b) Market factory to deploy new auction instances (c) MIS. This market place may be advertised through its portal to encourage small and medium clients to register as consumers or providers.

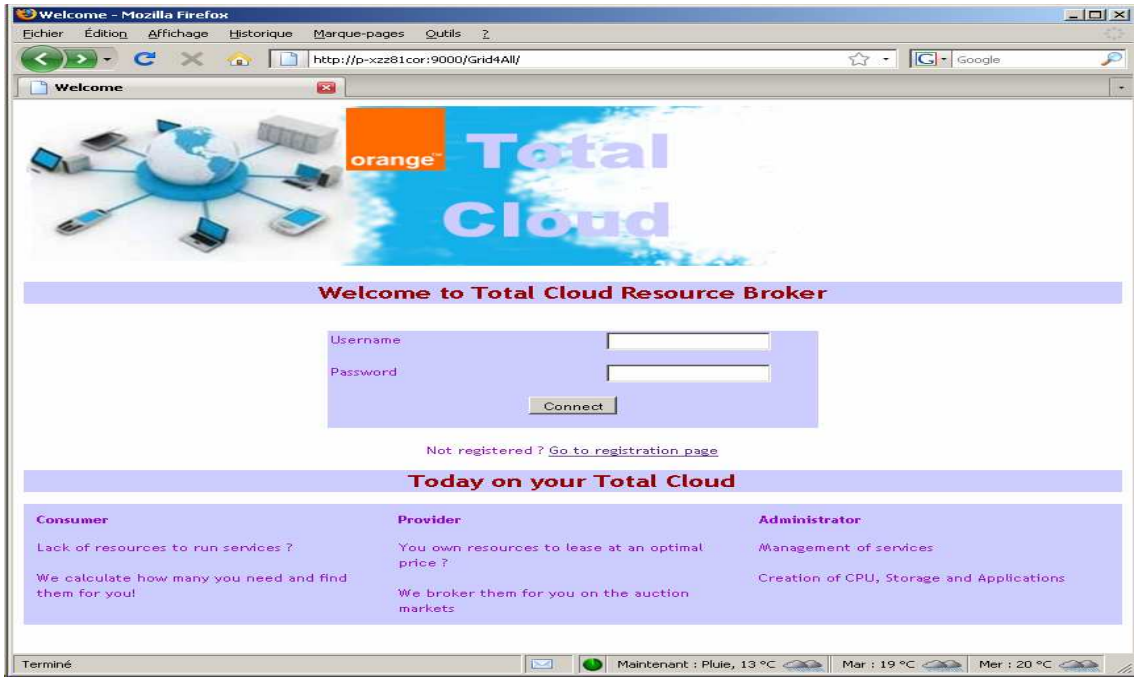


Figure 11 Portal to market place

The market place operator may provide specific tools for consumers and providers. Figure 11 shows interfaces offered to consumers. Consumers are allowed to execute one of a pre-configured set of applications. The consumer is expected to give the desired time horizon and budget; the system does the translation to generate adequate *Bids* that are forwarded to a suitable auction.

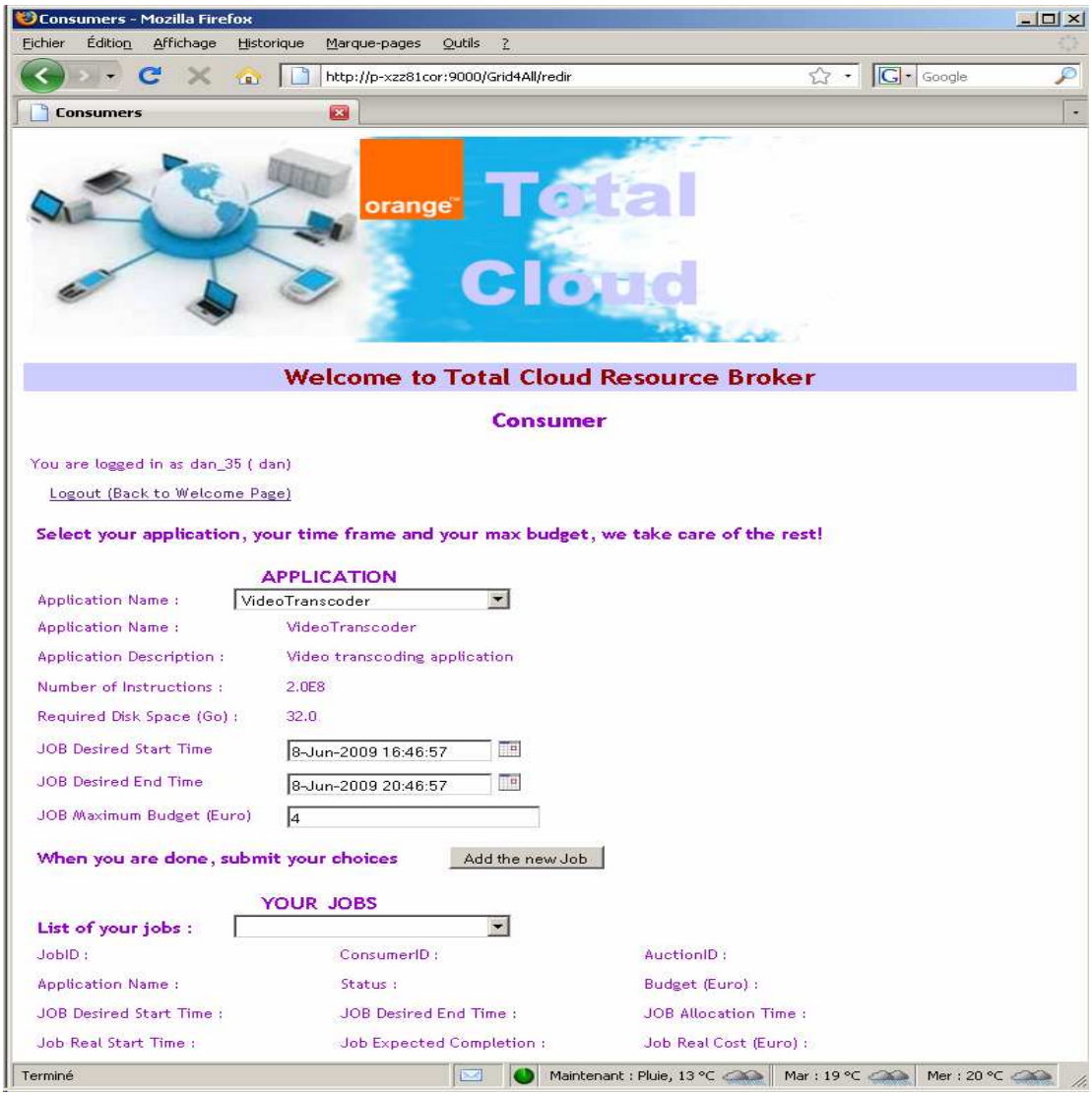


Figure 12 Consumer portal offered by market place

The MIS is designed to retrieve and provide information within large-scale market environment. Therefore, it can also be used in other on-line markets other than for computational resources. Besides using the MIS within markets, the flexible design of the implementation allows the usage for other Grid applications with a scalable nature for example as an information system for Virtual Resources.

6.12 Conclusions and lessons learnt

Small and medium scale companies are indeed turning to Cloud providers (Amazon's EC2 service is used by about 60,000 customers and generates revenues of around 130 million dollars). Nevertheless we do not have conclusive data and analysis on the needs from a Grid (Cloud) market place. It is not clear that Cloud computing will lead to end of resource ownership, even though the ratio of owned resources versus on-demand leased resources may shift.

Some questions are still to be answered, first technical and then economical. Cloud computing by itself has a number of open issues including interoperability. In this section we focus on market places. Important technical questions are (a) What appropriate pricing and allocation mechanisms, i.e. clearly a single mechanism is not apt for every kind of usage and every kind of ICT resource (b) How to monitor SLA and

ensure that guaranteed agreements are respected. In case of violations how to ensure that the end clients are protected and their applications continue to execute (c) How to incorporate networking resources.

From an economic perspective we need to ask (a) whether a grid economy necessary and viable, when does it become cheaper to lease resources than to buy resources (b) can the current situation where cloud computing may be characterized as a seller's market be changed through competition where every consumer will also sell its excess resources at the Grid market, thus stimulating competition and lowering prices. (c) Are edge resources compatible with Green IT?

From the technical side, a lesson learnt from the MIS is the need for real-time information. Simulations showed that there is a margin for imperfect data and data with latency in regard to equilibrated market prices. However, we have also seen that the time issue is quite important in P2P systems and can exceed, depending on the network, 10 seconds for 10000 participants. Therefore, we propose and evaluate optimizations in regard to faster routing structures and approximations.

For the future, we will evaluate the CAS with simple negotiation agents, which base their price calculation on the MIS. We will setup a tested of over 25 machines and deploy the mentioned components in Planetlab to have a real distributed test environment. The evaluation will compare trading with market information and without the MIS.

For a market to be functional essential platform services need to be integrated The market place needs to ensure that agreed contracts are enforced and in case of provider failures ensure that consumers do not suffer. It needs to provide intelligent tools for consumers and providers to plan their capacities and their on-demand allocations.

7 Overall result summary

7.1 Software prototypes and status

7.1.1 Semantic Information Service

Name of product	SIS (Semantic Information System)
Licence type	LGPL 3.0 with a written licence agreement
Source URL	http://ai-lab-server.aegean.gr/svn/ai-lab/sis/Service/src/
Software package download URL	http://icsd-ai-lab.aegean.gr:8080/grid4all_sis/downloads.jsp
Owner and credits	Ai-Lab, Dept. of Information and Communication Systems Eng. , University of the Aegean
Main features	Semantic-based advertisement and discovery of Grid resources and services. Matchmaking of suppliers who offer resources and consumer who request resources by providing them end point references to auction-markets that trade them (vice-versa).
Other major non-G4A software required	Jena, Pellet, Apache Axis
Main advantages	Semi-automatic tools to aid ontology/semantic technology unaware users and developers to benefit of a rich semantic-based discovery system.
Main disadvantages	Currently a centralized architecture has been implemented. A decentralized design is proposed

7.1.2 Selection Service

Name of product	SbQA
Licence type	LGPL
Source URL	http://sbqa.gforge.inria.fr
Software package download URL	http://sbqa.gforge.inria.fr
Owner and credits	INRIA, ATLAS team (Contributors: Philippe Lamarre, Jorge-Arnulfo Quiané-Ruiz, and Patrick Valduriez)
Main features	Architecture, innovative algorithm and API to select and rank query results in a personalized way.
Other major non-G4A software required	
Main advantages	Enables a registry and discovery service to select query results based on consumer, provider preferences and provider load.

Main disadvantages	Centralized service, currently updating of provider dynamic conditions such as load is not done.

7.1.3 Market information service

Name of product	MIS (Market Information Service)
Licence type	LGPL 3.0 with a written licence agreement
Source URL	https://code.ac.upc.edu/projects/dmis https://scm.gforge.inria.fr/svn/grid4all/g4a-wp2/GRIMP
Software package download URL	https://code.ac.upc.edu/projects/dmis https://scm.gforge.inria.fr/svn/grid4all/g4a-wp2/GRIMP
Owner and credits	Universitat Politechnica de Catalunya, Barcelona
Main features	Decentralized information system providing aggregated and summarized information on queries over published topics. Automatic uncertainty management to balance accuracy of information and the cost (time and number of messages). Provides specific support for distributed market places by providing information on prices, volume of supply and demand.
Other major non-G4A software required	Pastry, Kademlia, Scribe
Main advantages	Flexible architecture that may be adapted to different access technologies and that allows extensions with new matching and filtering algorithms. History maintenance.
Main disadvantages	No replication, hence data may be lost. This is planned for future work.

7.1.4 Configurable auction server and mechanisms

Name of product	Configurable auction server and auction mechanisms
Licence type	LGPL
Source URL	https://scm.gforge.inria.fr/svn/grid4all/g4a-wp2/GRIMP
Software package download URL	https://scm.gforge.inria.fr/svn/grid4all/g4a-wp2/GRIMP
Owner and credits	France Telecom and Universitat Politecnica de Catalunya, Barcelona
Main features	Perform auction services for complex combination of resource offers and requires through simple APIs. Flexible auction mechanisms to trade time-differentiate computational resources in combinatorial bundles or variable units of single resources.
Other major non-G4A	CPLEX may be used but it is not mandatory

software required	
Main advantages	Pricing mechanism that computes commodity prices as function of time. Flexible bidding language that may be used with a wide range of auction mechanisms. Architecture-based and simple to deploy.
Main disadvantages	Lack of tools to interpret market statistics, require support for consumer and provider decision making

8 Conclusions

This deliverable reports the overall status of resource management services and tools that have been designed and fully or partially developed within WP2. The objective of the project was to develop tools, middleware and services that enable construction and deployment of democratic grids, i.e. grids built by individuals and small organizations and using their own resources. This work package has contributed to this effort through its resource and application management tools and services.

The Grid4All platform is designed to minimize coupling between components; to enable reuse of components in situations other than where originally planned. At this stage of the project we have not completely integrated all developed tools to obtain a system that is ready to deploy on a test bed. Nevertheless the different components have been tested either unitarily or in simple client/server use cases.

The VO management system provides a set of comprehensive tools to deploy democratic grids. The project has focused research on deployment, but demonstration purposes, it has also implemented a generic set of management tools that includes a complete security infrastructure.

The G4A-SIS (Semantic Information System) is a completely standalone product even though the primary objective motivating its design was to have a semantic discovery service that is aware of Grid economy. G4A-SIS may be used in deployment scenarios beyond project needs.

We believe that we have produced interesting results in the scope of Grid4All market place that could be used in further research and/or development work in particular with the advent of Cloud computing. Cloud computing has caused profound changes in the way medium-sized enterprises perceive and plan their ICT infrastructure. This trend will catch up even to large enterprises, small organisations and even individuals. In this sense, Grid4All and its vision was quite pertinent, since it addressed this segment of the population. A realistic ICT market place that regulates supply and demand and offers incentives to suppliers and consumers to participate is still relevant even in this context.

9 References

[Deng05] G. Deng, J. Balasubramanian, W. Otte, D. C. Schmidt, and A. S. Gokhale. DAnCE: A QoS-Enabled Component Deployment and Configuration Engine. In *Component Deployment, 3rd Int. Working Conference, CD 2005*, number 3798 in Lecture Notes in Computer Science. Springer, 2005.

[Albrecht07] J. Albrecht, R. Braud, D. Dao, N. Topilski, C. Tuttle, A.C. Snoeren, and A. Vahdat. Remote Control: Distributed Application Configuration, Management, and Visualization with Plush. In Proceedings of the Twenty-first USENIX Large Installation System Administration Conference (LISA), November 2007.

[Flissi06] A. Flissi and P. Merle. A Generic Deployment Framework for Grid Computing and Distributed Applications. In *OTM Confederated International Conferences, Grid computing, high performAnce and Distributed Applications (GADA 2006)*, volume 4276 of *Lecture Notes in Computer Science*. Springer, 2006.

[Kon05] F. Kon, J. R. Marques, T. Yamane, R. H. Campbell, and M. D. Mickunas. Design, implementation, and performance of an automatic configuration service for distributed component systems. *Software - Practice and Experience*, 35(7), 2005.

[Lan05] L. Lan, G. Huang, L. Ma, M. Wang, H. Mei, L. Zhang, and Y. Chen. Architecture Based Deployment of Large-Scale Component Based Systems: The Tool and Principles. In *Component-Based Software Engineering, 8th International Symposium (CBSE)*, volume 3489 of *Lecture Notes in Computer Science*. Springer, 2005.

[Mikic02] M. Mikic-Rakic and N. Medvidovic. Architecture-Level Support for Software Component Deployment in Resource Constrained Environments. In *Proceedings of the IFIP/ACM Working Conference on Component Deployment (CD'02)*, number 2370 in Lecture Notes in Computer Science. Springer, 2002.

[Wang06] Wang, X., Li, W., Liu, H., and Xu, Z. 2006. A Language-based Approach to Service Deployment. In *Proceedings of the IEEE international Conference on Services Computing* (September 18 - 22, 2006). IEEE Computer Society, Washington, DC, 69-76.

[Goldsack03] P. Goldsack. SmarFrog: Configuration, Ignition and Management of Distributed Applications. Technical Report http://www-uk.hpl.hp.com/smartfrog, HP Research Labs, 2003.

[Valetto03] G. Valetto and G. E. Kaiser. Using Process Technology to Control and Coordinate Software Adaptation. In *25th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2003.

[Rowanhill07] J. C. Rowanhill, G. S. Wasson, Z. Hill, J. Basney, Y. Kiryakov, J. C. Knight, A. Nguyen-Tuong, A. S. Grimshaw, and M. Humphrey. Dynamic System-Wide Reconfiguration of Grid Deployments in Response to Intrusion Detections. In *High Performance Computing and Communications, 3rd International Conference, HPCC 2007*, number 4782 in Lecture Notes in Computer Science. Springer, 2007.

[Badonnel04] A. Keller and R. Badonnel. Automating the Provisioning of Application Services with the BPEL4WS Workflow Language. In *15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM*, number 3278 in Lecture Notes in Computer Science. Springer, 2004.

[Bastarrica98] M. C. Bastarrica, A. A. Shvartsman, and S. A. Demurjian. A Binary Integer Programming Model for Optimal Object Distribution. In *2nd Int. Conf. On Principles Of Distributed Systems. OPODIS 98*. Hermes, 1999.

[Malek05] S. Malek, M. Mikic-Rakic, and N. Medvidovic. A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems. In *Component Deployment, Third International Working Conference, CD 2005*, number 3798 in Lecture Notes in Computer Science. Springer, 2005.

[Tibermacine07]C. Tibermacine, D. Hoareau, and R. Kadri. Enforcing Architecture and Deployment Constraints of Distributed Component-Based Software. In *10th International Conference Fundamental*

Approaches to Software Engineering (FASE), volume 4422 of *Lecture Notes in Computer Science*. Springer, 2007.

[Mikic02] M. Mikic-Rakic and N. Medvidovic. Architecture-Level Support for Software Component Deployment in Resource Constrained Environments. In *Proceedings of the IFIP/ACM Working Conference on Component Deployment (CD'02)*, number 2370 in Lecture Notes in Computer Science. Springer, 2002.

[Arshad07] N. Arshad, D. Heimbigner, and A. L. Wolf. Deployment and dynamic reconfiguration planning for distributed software systems. *Software Quality Journal*, 15(3), 2007.

[Kichkaylo04] T. Kichkaylo and V. Karamcheti. Optimal Resource-Aware Deployment Planning for Component-Based Distributed Applications. In *13th International Symposium on High-Performance Distributed Computing (HPDC 2004)*. IEEE Computer Society, 2004.

[Heydarnoori06] A. Heydarnoori, F. Mavaddat, and F. Arbab. Towards an Automated Deployment Planner for Composition of Web Services as Software Components. *Electr. Notes Theor. Comput. Sci.*, 160, 2006.

[Lamparter & Schnizler, 2006] S. Lamparter and B. Schnizler. (2006) Trading Services in Ontology-driven Market'. In Proceedings of the 2006 ACM symposium on Applied Computing (SAC'06), Dijon, France.

[Ragone et al, 2007] A. Ragone, U. Straccia, T. D. Noia, E. D. Sciascio and F. M. Donini. (2007) Vague Knowledge-bases for Matchmaking in P2P E-Marketplaces', 4th European Semantic Web Conference (ESWC 2007), Springer-Verlag, LNCS 4519, pp. 414–428.

[Kunal et al, 2005] V. Kunal, K. Sivashanmugam, A. Sheth, A.Patil, S.Oundhakar, and J. Miller. Meteors wsdi: A scalable p2p infrastructure of registries for semantic publication and retrieval of web services. Inf. Technol. and Management, 6(1):17–39, 2005

[Babik & Hluchy, 2006] M. Babik and L. Hluchy. A Broker Based Architecture for Automated Retrieval and Invocation of Stateful Services. R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 204–211, 2006

[Bao et al, 2006] Bao J., Caragea D., Vasant Honavar, A Tableau-based Federated Reasoning Algorithm for Modular Ontologies, Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)

[Adjiman et al, 2006] Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset and Laurent Simon, **Distributed Reasoning in a Peer-to-Peer Setting: Application to the Semantic Web**. Journal of Artificial Intelligence Research, Vol. 25, pages 269-314, 2006

[Serafini & Tamilin, 2005] Luciano Serafini, Andrei Tamilin. "DRAGO: Distributed Reasoning Architecture for the Semantic Web". Proceedings of the Second European Semantic Web Conference, 2005

[Stuckenschmidt, 2001] Heiner Stuckenschmidt, Modularization of Ontologies IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, Del 21

[Soma & Prasanna, 2008] Soma, R.; Prasanna, V.K., "Parallel Inferencing for OWL Knowledge Bases", Parallel Processing, 2008

[Tatarinov et al, 2003] Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The piazza peer data management project. SIGMOD Record, 32(3), 2003.

[Nejdl et al, 2002] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: a p2p networking infrastructure based on rdf. In International World Wide Web Conference (WWW), 2002

[Aberer et al, 2004] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. van Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In International Semantic Web Conference (ISWC), 2004

[Vouros et al, 2008] G. A. Vouros, A. Papasalouros, K. Kotis, A. Valarakos, K. Tzonas, X. Vilajosana, R. Krishnaswamy, N. Amara-Hachmi (2008). The Grid4All ontology for the retrieval of traded resources in a market-oriented Grid, Special issue on "Web/Grid Information and Services Discovery and Management", Int. J. Web and Grid Services, 4(4):418-439

[Vouros et al, 2008b] G. A. Vouros, A. Valarakos, and K. Kotis (2008) Uncovering WSDL Specifications' Data Semantics. 2nd International Workshop on Service Matchmaking and Resource Retrieval (SMR2), ISWC 2008, Germany

[Serafini &Tamilin, 2005b] Luciano Serafini, Andrei Tamilin. "Distributed Instance Retrieval in Heterogeneous Ontologies". In Proceedings of SWAP 2005, CEUR Workshop Vol 166

[Schütt et al, 2008a] T. Schütt, M. Moser, S. Plantikow, F. Schintke, A. Reinefeld, A Transactional Scalable Distributed Data Store: Wikipedia on a DHT, 1st IEEE International Scalable Computing Challenge, SCALE 2008, Lyon, (first price award winning paper), May 2008

[Schütt et al, 2008b] T. Schütt, F. Schintke, A. Reinefeld, Scalaris: Reliable Transactional P2P Key/Value Store - Web 2.0 Hosting with Erlang and Java, 7th ACM SIGPLAN Erlang Workshop, Victoria, September 2008.

[Skoutas et al, 2008] D. Skoutas, D. Sacharidis, V. Kantere and T. Sellis, "Efficient Semantic Web Service Discovery in Centralized and P2P Environments", Proceedings of the 7th International Semantic Web Conference (ISWC 2008), Karlsruhe, Germany, October 2008.