Project no. 034567

Grid4All

Specific Targeted Research Project (STREP)

Thematic Priority 2: Information Society Technologies

D2.3 Specification & Initial prototype of Grid4All Resource Management

Due date of deliverable: 01-06-2008

Actual submission date: 11-07-2008

Start date of project: 1 June 2006 Duration: 36 months

Contributors : UPRC, UPC, FT, INRIA

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

Abbreviations used in this document

Abbreviation / acronym	Description

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 deliverable presents the design, specification and current implementation status of the Grid4All resource management system. The term *resource* essentially refers to computational and storage resources, where a single node may provide of course both the capabilities.

Core VO services that provide the essential management functionality within a VO are described in detail in the D2.2[1]. The decentralized discovery service allows applications to selectively discover currently available resources, where this selection is done by applying a filter on the properties describing resources. The membership service allows users to join and leave VOs, to search for other members and to associate attributes to members. Application components may be deployed on one or more nodes; this may be done either by using programmatic interfaces or by interpreting the declarative architecture descriptions. The Fractal Architecture Description Language is used to describe application architectures. The security system may be used to describe access control policies and implement policy decision and enforcement points.

Resource management described in this deliverable deals with execution management, resource brokering and service discovery. They use the aforementioned core VO services.

In Grid4All, Resource management is done at two levels: within the scope of a single VO and across VOs. Within a VO, core services allow applications to discover resources matching criteria and to provision these resources. Execution management is used to schedule tasks[2] and manage their execution. Data management described in D3.3 and D3.4 provide VO-wide file management services. Multiple VOs may exist on the Internet. The Semantic Information Service is provided for users and applications to discover resources and services. VOs may also allocate computational resources at the market-place.

This document describes the current implementation status, the programming interfaces, and the scenarios in which the provided functionality will be integrated to provide combined capabilities for end applications. We describe the interactions of the software components with each other for the purpose of jointly providing capabilities to users and applications. We conclude by outlining directions for future work.

[1] We recommend reading D2.2 before reading this deliverable.
[2] We focus on bag-of-tasks applications.

2 Introduction

This deliverable describes the current status of three major resource management services: Execution Management, Resource brokering and Semantic-based service discovery. To support execution of applications following the bag-of-tasks paradigm, execution management provides an offline schedule planner that is used to estimate application completion time. In conjunction with this, XtremWeb Desk-top computing middleware[3] is used to manage the run-time aspects such as starting of tasks, monitoring its status and gathering of results. This middleware has been partially adapted to leverage the deployment facilities offered by VO run-time software. Applications execute on resources currently available in the VO. On need, resources may be leased from resources markets. Elastic applications adjust to available resources at both deployment time and run time; the completion time will decrease if available computational capacity is increased. Offline scheduling permits estimation of completion times prior to execution and adjustment of the quantity of resources needed to meet timing requirements. If the VO does not have sufficient amount of resources, additional resources can be requested from the market.

Resources or more precisely leases to resources may be allocated at resource market-places. The Grid4All Market-Place (GRIMP) provides a set of services and tools to operate resource market-places. They can be accessed programmatically by resource consumers and providers. GRIMP addresses new categories of providers and consumers or resource. Workload characterisation and resource availability has been well analyzed within High Performance Computing production grids; this is not so in the case of Democratic Grids. We propose an evaluation scheme that defines basic parameters to characterize consumers and providers and captures the requests by simple utility functions.

Grids are seen as a ubiquitous utility for users and small organisations. Resources and services are deployed and exposed to the Grid users who offer and request them, in a market-oriented environment. In a market-oriented environment, resources are made available through spontaneous peer-initiated markets. To facilitate discovery and selection of auction-based markets, we develop the SIS, which provides the means to discover markets trading resources. Application services may also be advertised and discovered using the SIS. This chapter describes the architecture, APIs and the technological choices driving the implementation.

The rest of the document is structured as follows. Chapter 3 presents a short introduction to concepts of resource management. Execution management is presented in Chapter 4 and explained through support furnished for bag-of-tasks applications. A video transcoding application is used to illustrate the approach. Chapter 5 and 6 describe, respectively, the tools and services to operate resource market-places and the Semantic Information Service. Chapter 7 gives an illustration of how all the services fit together.

For the sake of visibility, these chapters are brief in their descriptions. The appendices provide more details of provided functionalities.

[3] www.xtremweb.net

3 Resource management

Computational and storage resources are managed and allocated to applications to deliver value to end users. The main purpose of resource management is to allocate and provision resources (CPU, storage, physical memory and network bandwidth) to applications. Resource management addresses:

- ➢ Resource discovery, i.e. identification and matching of services and resources within the system (A VO in our case), according to properties that characterize resources.
- ➢ Resource brokering, i.e. selection and decision to allocate a resource to a requesting application.
- ➢ Scheduling or planning to decide which task should execute on which resource and when.
- ➢ Deployment or installation of application software on target nodes and their configuration.
- ➢ Execution Management or run-time lifecycle management of application tasks until their completion. Execution management uses many of the previously described services to accomplish their function.

Access modules provide concrete means by which allocated resources are used to run applications. Within traditional Grid computing systems, resources are accessed using methods such as interfaces to batch queue systems, remote execution protocols such as *ssh*. Within Grid4All, users of computational resources are members of virtual organisations. Resources belonging (or leased) to a VOs are organized as an overlay network. The nodes of the overlay network execute management services and applications on behalf of users. New compute nodes join the VO by using the Grid4All protocol to *join* the overlay. Once joined, these may be discovered by applications and then used using the APIs as described in D2.2.

D2.2 has described the core VO services that provide the basic management services; deployment and basic discovery of resources within the VO. The current deliverable describes the higher level functionality and services; semantic based information services, resource brokering and scheduling.

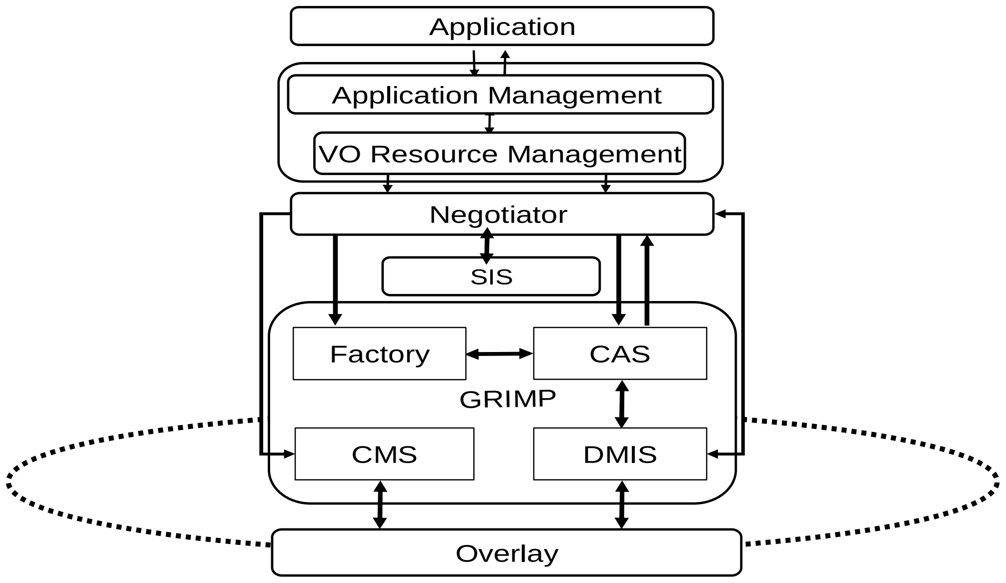
3.1 Relation between the different services

This section describes the relation between the different resource management services keeping in mind the application scenarios described within the D4.7. The scenario is in a context where an end application such as the gMovie, or the CNSE network simulator, is adapted to use the Grid4All APIs; to deploy application components and manage their execution. Applications are managed by using management services described within the deliverables D2.2 and D2.3. Management implies discovering and allocation of resources, deployment of application components, monitoring of tasks and adaptation to unexpected system events such as failures, leaves or joins of resources.

Figure 1[4] gives the overall layering of the different components at a high level of abstraction. At deployment of application, the minimum resources required to execute the application is evaluated. Application management code is expected to use the different services (discovery, allocation, task scheduling and dispatching) and select the candidate machines best suiting it. Run-time (self) management monitors current state of execution and takes decisions concerning the reorganisation of the application. D1.2 has described the Distributed Component Management System (DCMS). Self-managing applications may reliably execute on democratic grids when written using this framework.

Resource management services are described in more details in subsequent chapters.

[4] This figure does not give all the architectural layers, in particular the DCMS and overlay services.



5

Figure 1

3.2 Resource and service discovery revisited

Within Grid4All, there are two types of discovery services. D2.2 describes the basic resource discovery service and it's API. This deliverable presents the Semantic Information Service, which is used for service discovery. So why do we need two different services?

The basic resource discovery described in D2.2 provides a relatively simple mechanism to find nodes that are currently part of the VO. Such compute nodes may be of two origins. First, a user may login to the VO and contribute his/her computer. Second, compute nodes may be leased using the Grid4All resource marketplace. In either case, the compute nodes join the VO and the underlying overlay network and they have to be discovered by higher-level management. Compute nodes have a list of properties (CPU speed, memory size, storage space), and resource discovery is based on filtering on values of those properties. Once discovered, applications allocate computational and storage capacity on the nodes and deploy components. Deployed applications may register themselves within the naming service. The scope of the naming service is 'local' to a VO; i.e. can be looked up only by other applications executing within the VO.

The SIS is used to advertise and discover services across VOs. It has a wider scope and is not restricted to a single VO. It is currently designed as a centralized service that executes on its own resources at a well-known address. The SIS functionality may be accessed by applications executing on any VO. It is not our current objective to use this same SIS to discover the components and applications executing locally within the VO.

4 Execution management

This section describes the main Execution Management modules. The main services are scheduling (planning of application tasks) and execution of application tasks on resources. We give an overview of the relevant APIs and describe the scheduling heuristics. We present the architecture and conclude with a summary of the research conducted in the area of scheduling on large area networking environments.

4.1 Overview

The three main components of the Execution Management are the Scheduling Service, the XtremWeb Server and the XtremWeb Worker. XtremWeb components are responsible for execution of bag of tasks applications. The Scheduling Service computes an execution plan (assignment of tasks to resources). The makespan, i.e., the time at which the last task finishes execution, is extracted from the execution plan generated by the scheduler.

XtremWeb[5] is a middleware to build lightweight Desktop Grid that gathers unused resources of Desktop Computers (CPU, storage, network). Its primary features permit multi-users, multi-applications and cross-domains deployments. XtremWeb turns a set of volatile resources spread over a LAN or Internet into a runtime environment for highly parallel applications. This open source software will be used to prototype execution management in Grid4All. XtremWeb consists of the server part (XtremWeb Server) that distributes the tasks to its workers (XtremWeb Workers). The legacy *Worker* module is wrapped as a Fractal component and deployed using the core VO deployment service.

4.2 Scheduling heuristics and scheduler API

The scheduler assigns tasks to resources based on a suitable policy such as minimisation of makespan. When scheduling multiple tasks of a job, heuristics are used to find good makespans. MinMin heuristic is the default scheduler heuristic. It iterates by choosing the next task that has the minimum expected completion time (over all tasks). Offline scheduling and computing of worst case completion time is essential to estimate the minimum resources required for the execution of the application, taking into account the execution time and total budget given by the user.

The scheduler API is neutral to the used heuristic. Heuristics are engineered for special class of applications and require specific properties from the execution environment (e.g. for the data intensive application heuristic we designed in [SS-3], data should be distributed using a modified version of BitTorrent that gives predictable transfer completion times). The MinMin heuristic does not make assumptions about the execution environment (cf. [SS-5]). It is suitable for compute-intensive applications. Its complexity is O (nb_resources * (nb_tasks^2)). The input to scheduling is a set of tasks and a set of machines. A task is represented by its estimated completion time in terms of required computational capacity. A machine is represented by its capacity. The objective of the scheduler is to assign tasks to machines. The heuristic iterates over tasks and selects the task with minimum computation time and assigns it to the machine with resulting least aggregated completion time. These steps are repeated until all tasks are assigned.

The Grid4All scheduler can be used to find an estimate of the computation time, based on the specification of tasks and the resources that are available to execute the tasks.

[5] Software anterior to Grid4All and contributed to the project by INRIA.

```
Schedule findSchedule (Resource[] resourceSet,
                       Task[]     tasks,
                       int        schedulerType)
```

This function returns a plan for the tasks on the set of resources given as input, and uses the scheduling heuristic identified by the parameter “schedulerType”. The Resource object (read-only) describes the properties of a compute node and is described in Appendix C. We intentionally restrict the range of values of properties describing computational resources, to restrain the search space during request processing. Tasks are described by their main characteristics; the input data size, required deadline and an estimate of its required processing capacity. The output of the scheduler is a list of scheduled tasks reflecting the mapping of tasks to compute nodes. A scheduled task is a tuple: *Task*, *Resource* and Start time.

The gMovie transcodes video formats. A task in the context of this application corresponds to the fragment of the input video. We have designed a specific heuristic to schedule these tasks since we will use the pull scheduling mechanism implemented by XtremWeb. This heuristic is described in Appendix C .

4.3 Execution management

Execution management consists of three major parts: allocation of resources, planning of tasks on resources and execution of tasks on the allocated resources. Figure 2 gives the sequence of interactions between different VO management services. The gMovie application is built with a management layer that uses the resource management interfaces to allocate and execute the application. The Reservation Manager, described in detail in D2.2 is used by the gMovie management layer to allocate compute nodes. XtremWeb Desktop Computing middleware manages the execution of application tasks on the allocated nodes.

The slave components, i.e., the workers of XtremWeb, are deployed using the deployment service. This is done by gMovie management layer on notification of arrival of a new resource (from the Reservation Manager). The XtremWeb middleware is self-managing; it detects new joining workers. Workers pull in tasks from a queue maintained by the master. The Scheduler is used by the application manager to estimate the minimum quantity of compute nodes needed to transcode the video, within the user specified deadline. The gMovie management layer, represented by gMovieApplication in the figure, iterates with the Scheduler to obtain such estimates. The Scheduler gives the maximum completion time, given a description of compute nodes and a set of task descriptions. The gMovie manager iterates with the user until finding a satisfactory balance between completion time and the budget that the user is willing to spend. The user supplied budget is used to lease compute nodes at the resource market (described in Section 5). XtremWeb slave component is then deployed in each leased compute node (as and when they arrive).

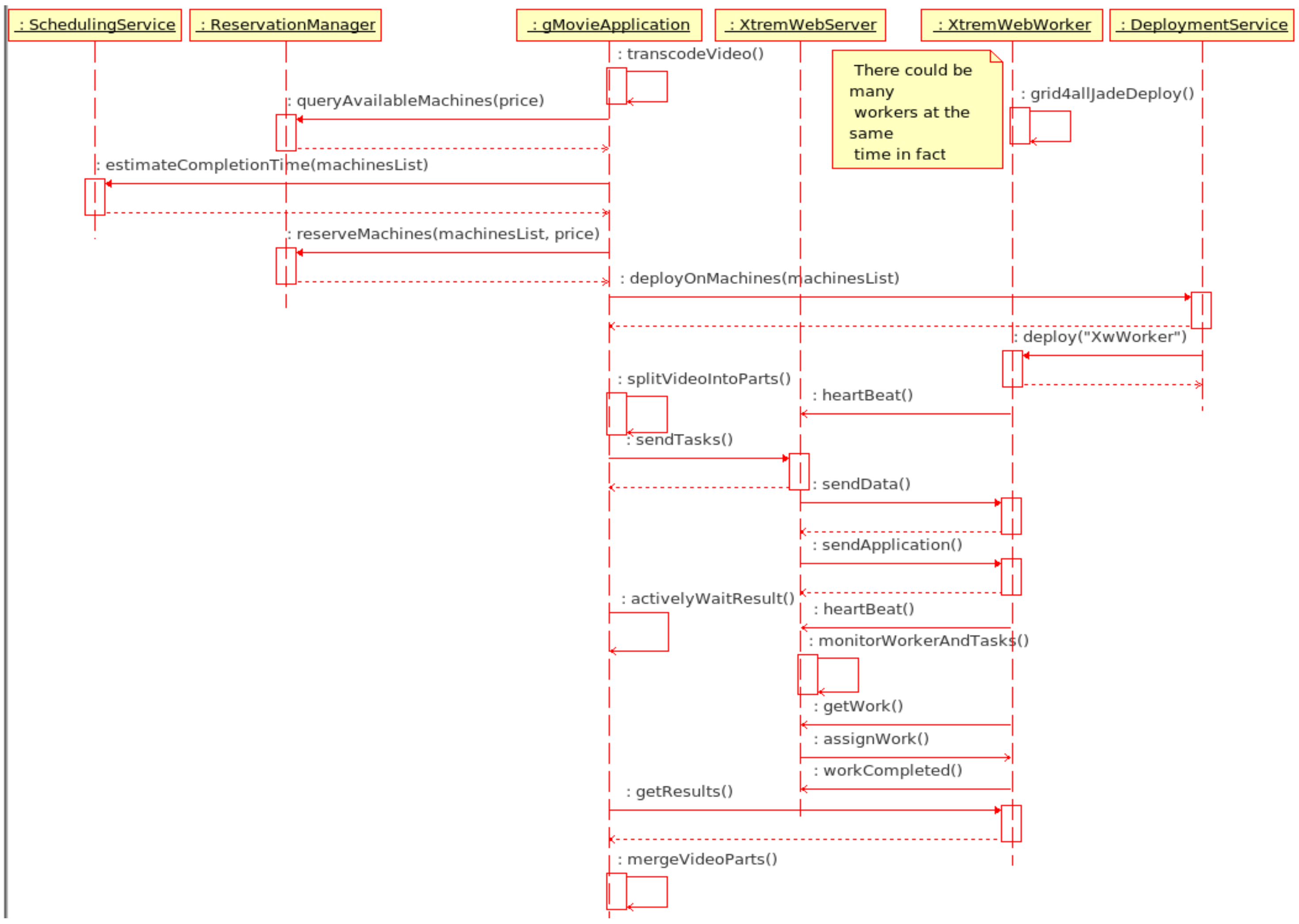


Figure 2 the gMovie demo using XtremWeb

4.4 Summary of Research Results

We present an overview of research conducted in the context of scheduling bag of tasks applications in desk top environments, similar to the environment of Democratic Grids targeted within Grid4All. Desktop Grids uses computing, network and storage resources from idle desktop PCs distributed over multiple-LANs, or the Internet, to compute a large variety of resource-demanding distributed applications.

While DGs offer a high return on investment, a critical issue is validation of results returned by participating hosts. Several mechanisms for result validation have been proposed, but the characterization of errors is poorly understood. To study error rates, we implemented and deployed a desktop grid application across several thousand hosts distributed over the Internet. We analyzed the results to give quantitative and empirical characterization of errors stemming from input or output (I/O) failures. We find that in practice, error rates are widespread across hosts but occur relatively infrequently. We find that error rates (a) tend to be stationary over time and (b) are not correlated between hosts. We evaluated state-of-the-art error detection mechanisms and describe the trade-offs for using each mechanism. We have reported this research in [SS-

2]. This result implies that scheduling heuristics must take in account error rates.

Desktop Grid applications access, compute, and store and circulate large volumes of data; In case of heterogeneous, large-scale and volatile environments, data management still mainly relies on ad-hoc solutions, and providing general approach is still a challenging issue. We have proposed the BitDew framework a programmable environment for automatic and transparent data management on computational Desktop Grids. BitDew [SS-4] relies on a specific set of meta-data to drive key data management operations, namely life cycle, distribution, placement, replication and fault-tolerance with a high level of abstraction. BitDew has distributed service architecture and integrates P2P components such as DHTs to implement distributed data catalog and collaborative transport protocols for data distribution. A high level of abstraction and transparency is obtained with a reasonable overhead, while offering the benefit of scalability, performance and fault tolerance with little programming effort.

Focusing on applications requiring large data transfers, in [SS-3], we demonstrated the advantages of using BitTorrent instead of FTP. Projects such as BOINC have confirmed our results. BitTorrent however suffers from drawbacks implying source modifications (its data transfer duration is too unpredictable) to either BitTorrent itself or to the scheduling heuristics.

Efficient data distribution is not the only factor that positively affects makespan. Throughput is important for multimedia applications. In [SS-1], we consider soft real-time applications executing on volatile resources where tasks arrive with well known order and rate. Resources are volatile and hence tasks scheduled on resources may miss their deadline if the resource leaves. In this study we show that the cumulative power of a DG follows a normal probability distribution. We therefore modeled the application by taking a buffer to hold intermediate results. Then we proposed to model failure rate (which can be seen as a throughput) as a function of this buffer's size.

Researchers from the community often ask themselves if it is more interesting to schedule data then computations, for applications where data transfer have a higher cost than computing. With volatile resources, heterogeneous and with variable communication capabilities, shared, etc. scheduling data is not only positioning them on resources but also managing them globally (lifetime, replication, migration, persistence). In order to study these topics, we developed a complete framework (described in [SS-4]) allowing management and distribution of data for a DG.

All of our studies, even with a model frame, are based on real-world observations (real availability traces, data transfer monitoring samples, etc.). For example, in [SS-2] the study is based on 10 millions of results computed by 35 000 hosts during 15 months.

5 Resource brokering

5.1 Introduction

Resources are computational and storage elements needed to execute applications. Brokering concerns selection and allocation of Grid resources. The Grid4All Market-Place (GRIMP) provides a set of tools and services to create resource market-places.

Members of VOs contribute their resources and use this shared pool. VOs may also incorporate resources from non-members; such resources can be leased through resource brokers that select and match consumer requests with supplier offers. When there are fluctuations in supply and demand we need mechanisms to arbitrate between requests and offers. Priority based, proportional sharing allocations are pertinent when consumers (applications or users requiring resources), belong to the same organisation. This is not the case when applications from multiple independent VOs contend for resources. Market based brokering with pricing mechanisms provide fair arbitration, gives incentives and is decentralized. The GRIMP (Grid4All Market-Place) provides services and tools to:

- Select suitable resource providers,
- Provide feedback from market to aid traders in brokering and negotiation,
- Mechanisms to allocate resources and establish prices of resources,
- Protocol to establish agreements between consumers and providers, including payment.

In systems with a small number of large providers (and similarly for consumers), negotiations could be bi-lateral. It is sufficient to provide discovery services that allow consumers to discover and select providers based on price, load and reputation. However when providers and consumers may be any actor on the Internet, this architecture does not scale. Hence in Grid4All, computational resources are allocated as *anonymous* entities at auction-based markets; consumers (providers) do not directly negotiate with resource providers (consumers), but through spontaneously instantiated markets.

From a consumer point of view the need is to find at a good price, and generally within a specific time frame, a bundle of resources composed with a certain (minimum) amount of processing, storage, service and network. If “10 CPUs” with 2 Gbytes are required, it is irrelevant if these are procured from one or multiple providers. The resources traded in the Grid4all are time-limited leases of computational resources represented by their:

- Processing capacity (CPU) and physical memory,
- Storage capacity and throughput,
- Network bandwidth (currently not supported).

Time-limited means that computational resources are allocated for specific intervals of time. This document will use the term *lease* to refer to such time-limited allocations of computational resources. Mature virtualisation technologies such Xen[6], VMware[7] provide isolation techniques to partition, isolate and share resources; a lightweight virtual machine can be the unit of allocation. Hence, we have adopted *non-shared* (in the classical sense of time-sharing), leasing of computational resources, i.e., a single compute node (indifferently virtual[8] or physical), is allocated at any time to only one consumer (VO). Computational resources are considered as non-divisible and allocated entirely for a specified time-interval.

Acquired resources should be accessible by VO members and their applications. Access methods range from remote execution protocols such as ssh to job submissions at batch queuing systems (as Grid services). We use protocols similar to those of peer-to-peer application overlay networks. Allocated nodes join the overlay network of the hosting VO; *join* and *leave* handling support offered by the core VO management is used to provide access to leased resources.

[6] http://www.xen.org/
[7] http://www.vmware.com/virtualization/
[8] Current prototype does not experiment with virtual nodes.

Acquired nodes join and leave the leasing VO using core VO functionality (specifically, the support for inviting remote nodes to the VO described in D2.2). The resource provider is a leasing authority. On acceptance of a leasing agreement, it is expected to configure the compute node that it has allocated for the lease such that the node joins the VO to which it has been leased. Compute nodes that are offered for leasing must be installed with the Grid4All container software and VO security components. After successfully joining the VO, the node can be discovered by VO members and is available for deploying applications. The hosting VO is expected to release the compute nodes on lease expiration.

D2.2 describes the Reservation Manager, the module that provides allocation services to applications. This module in conjunction with the Negotiator described in section 5.4.5 allocates resources using the market-place services. As a result of successful brokering, when a leased node joins a VO, we need to decide to which application this new resource should be provisioned. In our current prototype, there is one instance of Reservation Manager for each application. The leased nodes are invited to join the VO by the RM through the "remote addition" service that is described in D2.2.

It is clear that both the resource provider and the resource consumer may cheat. We do not address such behaviour.

5.2 Overall architecture and main components

We describe the main functionalities offered by the market-place. The term service and component is used in an interchangeable way. The term *service* (or *component*) is used to represent an entity that can be interacted with in a request-response manner. A component offers one or more interfaces through which its services can be accessed. Components are written in Java and clients access the service using a Java RMI stub or through client side libraries (which themselves access the remote objects). The market-place services themselves execute on a *special* VO with its own compute nodes.

Market Factory

This service allows registered participants of the market-place to select auction formats and to deploy the selected auction format on a compute node. The factory maintains a repository of executable auction formats that can be selected using the Market Description Language. Selected auction formats can then be deployed to instantiate a new auction server.

Distributed Market Information Service (DMIS)

This publish/subscribe service is used by market-place participants to disseminate and obtain information revealing market situation, e.g. prices of resources (CPUs, storage), aggregated supply and demand of resources. Information is published at the DMIS by auctions either during their execution or at their completion and is disseminated to clients who may either query or subscribe for notifications. The distributed market information service also provides aggregated and summarized information, i.e. it maintains historic data over time and space.

Configurable auction server (CAS)

Participants in computing markets may be: users or resource consumers, owners or resource providers and brokers that mediate between them. Consumers and providers interact through auction-based markets. To facilitate mediation of different kinds of resources, the auction server has been designed as a set of components implementing auction protocols and that may be instantiated using the Market Factory. Instantiated auctions may be configured using the APIs of the auction server. They are published at the semantic information service and discovered by traders that need to buy or sell computational resources. Participants use the APIs of the server to register and negotiate.

Currency Management System (CMS)

This service maintains user accounts and keeps tracks of user consumption by storing a log with each transaction. It is based on a probabilistic transactional mechanism built upon a structured peer-to-peer overlay. This component is meant to control and regulate economic transactions.

Negotiator

Market-place clients are consumers and providers who wish to sell or acquire computational resource leases. These are collectively referred to as negotiators. Negotiators use the APIs provided by the market-place services: to create auctions, obtain market information, participate at auctions, use the currency system for payments, and finally access (provide access to) the traded resources. Both types – consumers and providers -- of negotiators should implement a set of interfaces that are described in later sections. The market-place services expect that the clients implement appropriate *callback* interfaces on which the service calls back clients. A VO is expected to have at least one active instance of Negotiator if it has been configured to reserve/allocate resource leases at markets.

Different implementation issues (e.g. interfacing, providing security and authentication, providing functionality) have been decoupled into different sub-systems, enhancing modularity and easing future implementations of more advanced features. The GRIMP modules rely on the basic middleware services provided by WP1 and as described within the deliverables D1.2 and D2.2. These hide the heterogeneity and distribution of the platform on which they execute. Fractal component model is used to develop the main services and tools, for the following reasons:

- ➢ To provide an abstract architecture for auctions, enabling reuse and facilitating design of new auction formats.
- ➢ To leverage the DCMS and the core VO services to deploy and manage the components of the market-place.

5.3 Relation between the GRIMP modules

The main interactions between the DMIS, CMS, CAS, Market Factory and the Negotiator modules are described in this section. DMIS is a decentralized service and provides client side interfaces to the Negotiator that queries or subscribes for events, and the market, which is an instance of CAS that publishes events, e.g. clearing market prices, number of market participants. SIS is a centralized service that provides a set of Java APIs to publish information about instantiated auction-markets and to query/select one or more of these markets. While DMIS is a pub/sub service that disseminates dynamic market information; SIS is a registry of executing markets. The Negotiator – a software agent trading on behalf users and applications is the focus of most of the interactions. This agent is the principal client (or user of the market-place services; The Reservation Manager described in D2.2 requests the Negotiator to allocate computational resources.

The Negotiator (buyer role) executes the following steps on reception of a resource request:

- ➢ Decide when to negotiate based on current and historical market information and the time and price constraints specified in the request. The DMIS provides such information when queried.
- ➢ Select the auction type that is best suits the resource request. If multiple types (and quantities) of resources are required for a rigid application (the complete requested bundle should be allocated), a combinatorial auction is chosen. The K-DA auction type is selected for elastic applications.
- ➢ Query the SIS and select from currently running auctions in the marketplace, one instance, which is the most appropriate to the current request. Section 6 describes in detail this process.
- ➢ Specify the bid and its parameters:
    - o Quantity of resources and their quality attributes, e.g. CPU speed or storage throughput;
    - o The time specification including the earliest lease starting time, the latest lease ending time, and the duration;
    - o The maximum price that the requester is willing to pay taking into account the importance of the request and the current market price.

- Register and participate at one of the selected auctions. The negotiator may participate in more than one independent auction, but it is up to the negotiator to ensure that it does not win the same request at more than one auction[9].
- Wait termination of auction and if successful collect the set of allocated leases.

The successful Negotiators obtain a set of leases in objects called *Agreements*. An Agreement is an object that encapsulates the references (URL) to the *AgreementProvider* and the *AgreementConsumer*, the type of resources (by their qualifying attributes), the number of units of resources, the times at which the resources are leased and the price of the transaction. The Negotiator at the consumer side (*AgreementConsumer*) redeems the allocated leases by contacting the provider through the *AgreementProvider* interface. The provider is expected to select the resource units, configure them and return the URLs of the selected resource units. It is to be noted that resources are characterized by their attributes (currently we allow CPU speed, memory size, storage size, network location) and resource units matching the required attributes are presumed to be interchangeable. Hence at time of negotiation at the Market, it is not necessary that the providers specify the exact physical machine (represented for example by its world wide name such as an IP address).

The Negotiator returns URLs of leased resources to the requesting Reservation Manager, which then requests the resources to join its overlay (when lease should start). Each join triggers a join event at the hosting VO at the lowest level (overlay) and is propagated to higher level handlers implemented by the resource managers.

We have prototyped the main functional units (SIS, CMS, DMIS, CAS) and specified the interfaces for the Reservation Manager (described in D2.2), the Factory and the Negotiator. We have started implementing the *ReservationManager* and the Negotiator tailoring them to the needs of bag-of-tasks applications. The section 5.6 presents the detailed design of the Negotiator. The *ReservationManager* uses the Negotiator to implement reservation and allocation of leases.

5.4 Design and implementation

5.4.1 Distributed Market Information Service

A challenge for a decentralized market information system is to meet the economic requirements in combination with the technical requirements of a distributed system. Aggregated and individual data such as prices, levels of supply and demand should be provided in near real-time. The technical realization has to cope with high churn and to scale with the number of traders and offers.

The DMIS architecture [MIS-6] consists of three layers shown in Figure 3: Market Information System (MIS) Application Layer, DMIS and Advanced Routing. Each layer addresses different technical or economic requirements. The MIS layer offers interfaces to Negotiators in Virtual Organization (VO) and provides the security and anonymization service. The Advanced Routing uses the DHT (to store subscriptions and topics) and KBR (Key-Based Routing) services to efficiently disseminate events; efficiency is measured in number of required messages. The additional services are aggregation, filtering, subscription and multicast. The Communication Layer uses the DHT and KBR, for large-scale scalability and the robustness.

[9] Currently a submitted bid may not be withdrawn. However even so, bids may not be withdrawn once the auction has computed allocations.

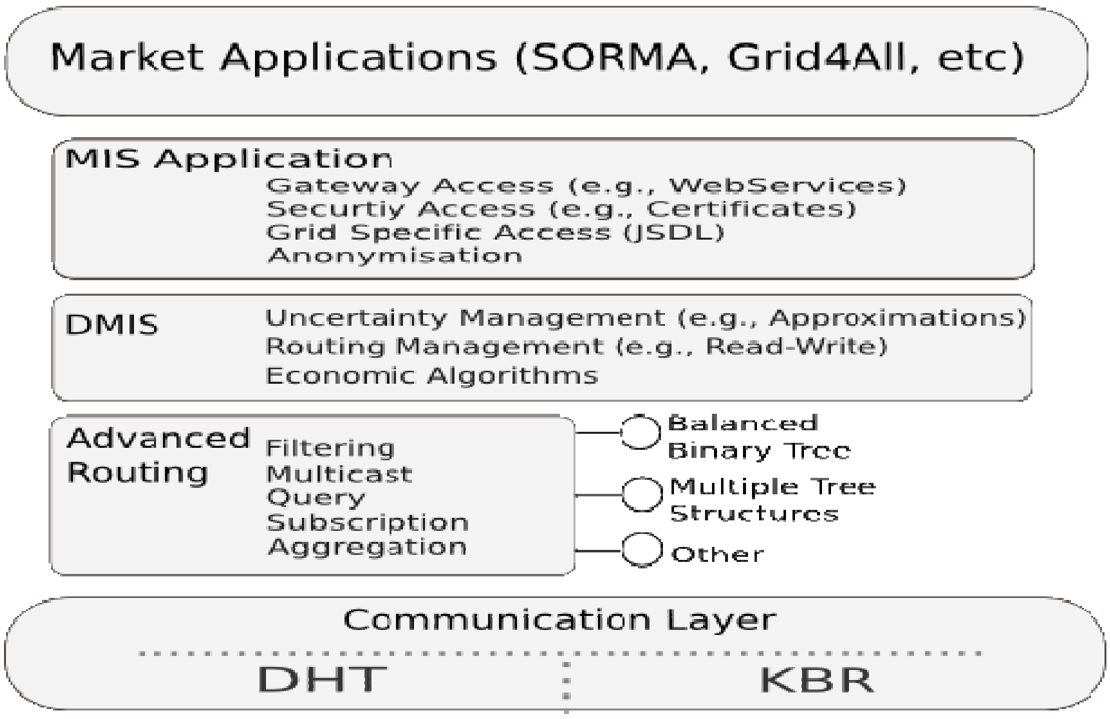

Market Applications (SORMA, Grid4All, etc)
MIS Application
Gateway Access (e.g., WebServices)
Securtiy Access (e.g., Certificates)
Grid Specific Access (JSDL)
Anonymisation
DMIS
Uncertainty Management (e.g., Approximations)
Routing Management (e.g., Read-Write)
Economic Algorithms
Advanced Routing
Filtering
Multicast
Query
Subscription
Aggregation
Balanced Binary Tree
Multiple Tree Structures
Other
Communication Layer
DHT
KBR

Figure 3 DMIS architecture layers

The **communication layer** has been implemented with both the Kademlia-based DHT [MIS-3] and Pastry [MIS-5] to show the flexibility of the architecture. The current prototype uses the existing open-source Scribe publish/subscribe system. To easily integrate with different overlay networks, the DMIS routing structures use standard APIs: *send*, *sendDirect*, *receive*, *put* and *get* [MIS-2].

The **Advanced Routing Layer** implements core functionalities listed below. It implements different aggregation tree structures as plug-ins. The ability to choose different tree (routing) structures permits higher layers to optimize the information provision process. The main functionalities are:

- **Filter-based routing:** Messages are forwarded to nodes which are interested in it; e.g., *an event publishing a resource price of 5 is sent to nodes interested in resources with a price lower than 4*.

- **Multicast:** Sends messages point-to-point to a subgroup of nodes in cases where a node knows all members of a topic. This will be changed to a tree-based propagation algorithm where a new node takes a certain place in the tree and knows only the direct parent(s) and children.

- **Query:** This function enables to execute a query for a *read-dominated* value within the marketplace. It follows an epidemic structure, binary-tree structure or multi-tree structure [MIS-1]. The current implementation organizes nodes as a binary-tree. Queries are propagated along this tree. Future work will implement multiple-trees, for improved robustness at cost of increasing number of messages.

- **Subscription:** This is the process to join a certain topic or content, and accordingly to obtain interested information. The current implementation allows subscriptions to topics. We are evaluating replacing this with content-based subscription.

- **Aggregation:** Provides summary information such as maximum, minimum, total and average. Aggregation improves scalability by reducing number of events to propagate. For complex aggregations such as averages, each node forwards the total towards its parents. Knowing the total number of children the average is calculated. Even more complex queries are a combination of more parameters (*select price where storage > 100 GB and memory > 3 GHz*).

The **DMIS layer** coordinates core querying, subscription and publishing functionalities for the client. It provides handlers, *SubscriptionHandler*, *QueryHandler* or *RequestHandler* for messages returning or entering the trader. The trader can invoke API methods to subscribe, query or publish [MIS-4]. Currently we store subscriptions at rendezvous nodes within the DHT and published events on nodes emitting the events. In the future, these will also be stored in the DHT. The published events are maintained over a time horizon.

The **MIS Application Layer** provides a flexible interface to the DMIS functionality and presents an adapter to the DMIS services. Web Services (in SORMA) and Fractal interfaces (in Grid4All) have been developed. This gateway access establishes the connection to clients executing on nodes within a VO. Security will be handled in this section via certificates.

The main programming interfaces are described below. Clients should implement notification handlers. Events encapsulate the market information to be transferred to interested traders. Events may be filtered by setting Patterns.

- **public boolean query (QueryHandler handler, Pattern pattern, long timeout) throws DMISException;** calling this method executes a request for a value like the price in the DMIS. The result is sent to the notify method of the assigned QueryHandler. The pattern specifies constraints (e.g. price < 100) that act as filters and an aggregation operator (minimum, maximum, average). The timeout defines the maximum duration of a query. DMISException is raised when the duration is exceeded. The method returns false if the query already exists.
- **public boolean subscribe (SubscriptionHandler handler, Pattern pattern) throws DMISException;** this method allows clients to subscribe to a topic. The pattern describes the events in which a client is interested. Published events in the matching topic are notified to the SubscriptionHandler, if the events match the specified filter.
- **public boolean unsubscribe (SubscriptionHandler handler, Pattern pattern) throws DMISException;** identified by the handler and pattern, the trader or participant will be unsubscribed from the content subscription.
- **public void publish (Event event) throws DMISException;** Clients publish events to the DMIS using this method. Events are transferred to interested subscribers.

The Appendix B.4 to this document presents the interactions between the main actors using the DMIS.

5.4.2 Auction Server

This section describes the Fractal based Configurable Auction Server (CAS). Auction is a process that implements rules to govern registration, bidding, pricing and determination of winners. Based on previously established taxonomies[10], we propose an approach using components to encapsulate auction activities and algorithms. A coherent set of components implementing a specific mechanism is described using the Fractal Architecture Description Language. The main motivations are:

- ➢ Configurability: Auction-based markets have a large number of configurable parameters; the *items* (and their configurable attributes) traded at the market, the different time-outs regulating the behaviour and scheduling, the control on number of registered participants etc.
- ➢ Reuse and extensions: Two auction types may be similar in almost all rules but a few; e.g., different pricing policies may be used even though bidding and allocation rules are the same. Moreover system and platform specific concerns such as deployment, configuration, registration and process control should be separated from auction specific rules programming.
- ➢ Deployment: On-demand creation of markets requires functionalities to facilitate deployment. D2.2 has described deployment of applications described using Fractal ADL.
- ➢ Distribution: Scalability and failure resilience are two key aspects that are addressed by means of distribution. A component model enables the execution of components at different locations that makes the framework more resilient to possible failures.

[10] Montreal Taxonomy

- Supporting multiple auction mechanisms: Auctions adhere to the maxim: one shoe does not fit all. The type of auction may depend on a number of variables such as the composition of request/offers, the time constraints, the privacy constraints, the allocation constraints etc.

Main messages and data

A **Participant** at an auction-market may be either a *seller* or a *buyer*. This object represents a negotiating agent that has a unique identifier across VOs. A participant must register at an auction before being able to access its service. A Negotiator may become a participant at an auction.

Bids encode the requirements of buyers and sellers. Bidding is the process of communicating the requirements and constraints of buyers and sellers to the auction. A bid is a logical expression that is represented as a tree with interior nodes representing logical operators (*OR*, *XOR*, and *AND*) and leaf-nodes to specify the bidder's request (or offer). XOR operators allow expressing substitute bids, i.e., the buyer is willing to accept exclusively one of the multiple options. OR bids indicate that the auction may accept any non-overlapping subset of the bid and that pricing is additive. AND nodes indicate that all leaf-nodes should be allocated. Bids are specified using XML encoded schema representing the resources required (offered) by buyers (sellers). Leaf-nodes embed the required (offered) item. Two types of resources are supported:

- Computational resource described by CPU speed, memory size and number of CPU units.
- Storage resource represented by size, disk throughput and number of storage units.

Basic resources can be combined into Aggregates (multiple units of similar resource) or Composites (bundles of different resource types). Auctions can be configured to trade basic, aggregate (CPU in sets of 8) or composite (3 CPUs and 40 giga of storage) resources. Leaf-nodes specify lease times (i.e. start time, end time and duration), prices, quantities and allocation constraints. Leaf-nodes may be imprecisely specified. For example, requests for 2 hours of CPU between 10:00 and 18:00 hours of a specific day could be expressed, without expanding to all possible combinations.

An **Agreement** object represents successful transactions buyers and sellers. Each allocation decided at an auction generates an Agreement encapsulating: the resource types, the price of transaction, the lease specifications and the partner information. A request from a buyer may be satisfied by more than one seller. Successful negotiators receive an *AgreementType* grouping all the allocated Agreements.

Main components and their management

The auction design space has been extensively studied. Similarities and differences of different mechanisms are well documented. Exploiting this, we have designed the CAS as a set of Fractal components. The architecture of the auction server is described with the Fractal ADL. Specific auction formats or types can be assembled by selecting the required implementation code. The design takes into account two aspects:

- Rules and algorithms representing the functional elements: Each component has a specified role and corresponds to a specific functionality within an auction.
- Dynamic or the process view: The execution of the auction process workflow follows the required set of interactions between the components (representing functional elements) and conforms to a given configuration of the auction.

Static view

Figure 4 represents the main components in the architecture. Components may be co-located on one single machine or placed on multiple nodes by changing declaratively the directives in the ADL. The capabilities of a component are accessed through the interfaces it *provides* and a component may only use the functions accessible through its *required* interfaces.

- **Bid management** encapsulates rules governing bidding. Incoming bids are pre-processed and validated for conformance. It implements data structures and algorithms to organize accepted bids.

- **Winner determination** component clears the auction. It labels bids as winners or losers; matches winning bids (one from seller and one from buyer). An objective function – typically social welfare or revenue maximization drives this matching. Clearing an auction is a hard problem particularly in the case of combinatorial auctions. Heuristic search-based algorithms may be implemented for specific cases of winner determination.
- **Pricing** component implements a specific pricing policy. The K-DA implements k-pricing policy; the transaction price is a weighted average of the *asking* and *bid* values.

The *Market* super-component includes the previously described *Auction* component and the following:

- **TradeInfo** component manages descriptions of *items* traded at a specific instance of the Market and is configured at creation. Its query interface informs participants of items traded and its configuration interface allows items to be configured.
- **Registration** manages the authentication and authorization of participants.
- **Feedback** provides publish/subscribe functionalities to subscribe to both market events like Quotes, and system events such as Termination. It generates and sends the *Agreement* to winning participants.

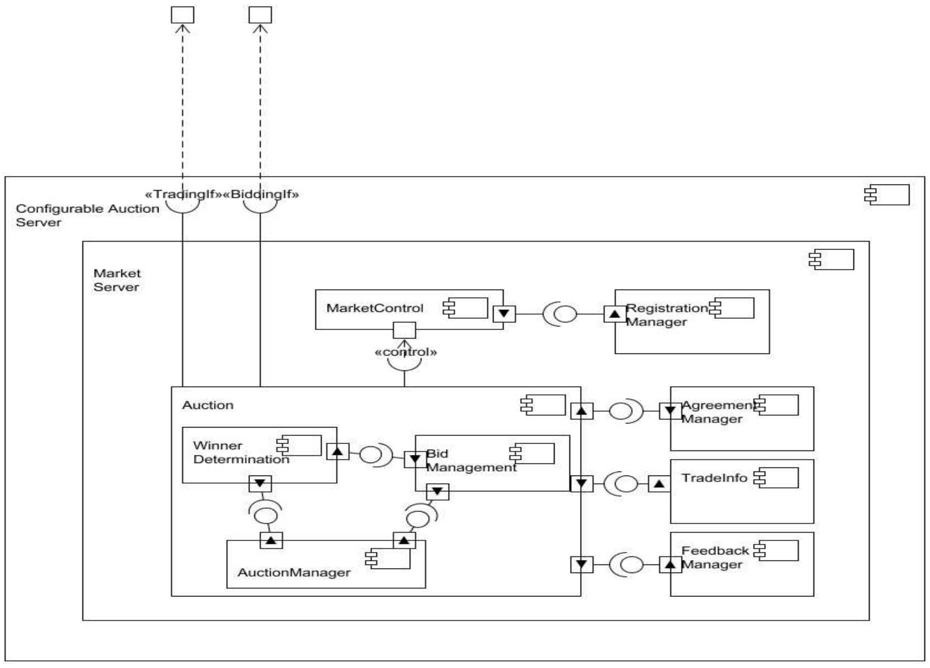


Figure 4 Auction server detailed architecture

Dynamic view

The dynamic view describes control and life-cycle management. The market process can be seen as an event-driven workflow. Events are generated by timers, method invocations, end of data transformations and synchronization points in the auction workflow. Transitions occur in the state of each component, the control flow is driven by external events and timers. Wiring of all possible execution paths is complex, given the number of configurable parameters in the design space[11]. Considering all possible execution paths result in a large number of possible workflows. Typical patterns are listed below:

- Sequential: Auctions may be configured to trigger clearing activity when bidding completes. But this may not always be the case; in continuous mode, matching is performed at each bid arrival.

[11] In D2.1, we have described the complexity of the auction design space.

- Parallel and concurrent executions: Auctions may be configured to allow registration to progress in parallel with bidding activities; i.e., one thread may be processing a bid from one client, while another thread may be registering a new client.
- Conditional: Registration activity may begin only when both the market is opened and the registration has been enabled. If both conditions are not true, then registration fails.
- Loop: In iterative auctions, auctions execute in rounds the same work flow pattern; accept bids, clear, send feedback.
- Event-condition-action triggers: For example, on reception of a valid bid, trigger clearing algorithm.

The auction (or market) process may be modelled as a hierarchical state machine; each contained component has its own state machine and is also subject to the state changes of the containing component. Components interactions can be modelled as a workflow orchestrating interactions of these state machines. For e.g., *Registration* component should accept registration requests only when *Market* is open. *Registration* may itself be configured to accept registrations only within a registration interval; both conditions need to be true to allow registrations. Similarly *Registration* may itself close, even if *Market* is still running, if end of registration has been triggered. Hierarchical and composite components can be viewed as managing composite states; child components are regions with their sub-states. Mapping this to semantics of hierarchical state machines is however non-trivial.

Analysis of state-of-art technology did not provide off-the-shelf design solutions:

- BPEL4WS: Even though promising and moreover recommended by SCA (Service Component Architecture[12]), BPEL4WS is heavy-weight; furthermore components are expected to be bridged through Web Services. With BPEL4WS, formal design methods such as Statecharts may be used to model and design auctions; then design mapping algorithms that generate BPEL4WS processes. The emergence of Fractal tools to bridge components and Web Services makes this approach a practical option.
- UML Hierarchical state machines: This is promising, however non-trivial to integrate with component-based architectures even though UML-2 supports component architectures. Components interact and synchronize through well-defined interfaces, whereas HSM model uses event-based state machine. There is also a lack of suitable runtime engines. Recent research projects such as [CAS-6] propose mapping of Fractal architectural and behaviour features within UML 2.0[13]. We may consider this approach at a later stage.
- Windows state machine workflow management: This moves the design space away from the component based approach that we have taken.

Currently, we model workflows manually (i.e. without using tools such as UML) and propose a native solution to control and synchronize components. Auctions are broadly single-shot or iterative. At single-shot auctions, participants may send their bid only once; the auction may clear immediately or at a scheduled time. Single-shot auctions preclude use of feedback and price discovery such that buyers may focus their bids on the most pertinent subset of traded items. When auctioning resource leases, feedback aids consumers to adjust deadlines and aggregated requested computational capacity. Iterative auctions evolve in rounds where each round executes the basic auction activities.

In Appendix B.1 of this document we describe the implementation of the generic server and specifically two auction mechanisms; K-pricing double auction to trade leases of single type of resources and the combinatory auction to trade bundles of computational resources. The CA has been specifically designed for Grid4All where typically providers own small quantities of resources, at least much smaller then the typical quantity that is requested by consumers.

[12] SCA is a set of specifications which describe a model to build applications using a Service Oriented Architecture. Application code is designed as a set of components which offer their capabilities through service-oriented interfaces (www.osoa.org)

[13] UML 2.0 http://www.omg.org

The K-pricing double auction is implemented in conformance to the CAS architecture. The CA has been formulated and implemented natively using CPLEX API. Our future work for combinatorial auctions are:

- Pricing model and heuristics based algorithms to compute (approximate) item prices,
- Implementation (or adaptation) of the CA model to within the Fractal based auction server. The main software engineering issue is designing the abstractions for the optimization model formulation of the combinatorial auction.

5.4.3 Currency management service

Currency Management System (CMS) is a distributed banking service which keeps track of user's consumption and contribution by storing user's balances and their transaction history in participant's accounts. Its main responsibility is to reliably store these accounts by means of a distributed and scalable storage system.

These accounts must be set up by users in order to participate in market transactions as each transaction will be credited through this service. CMS provides a simple API to open and close accounts as well as to deposit and withdraw G4A virtual currency against real money. Besides, it provides operations to transfer virtual currency from one account to another in the face of an economic transaction. Such transactions will be bound to an agreement reached through the CAS.

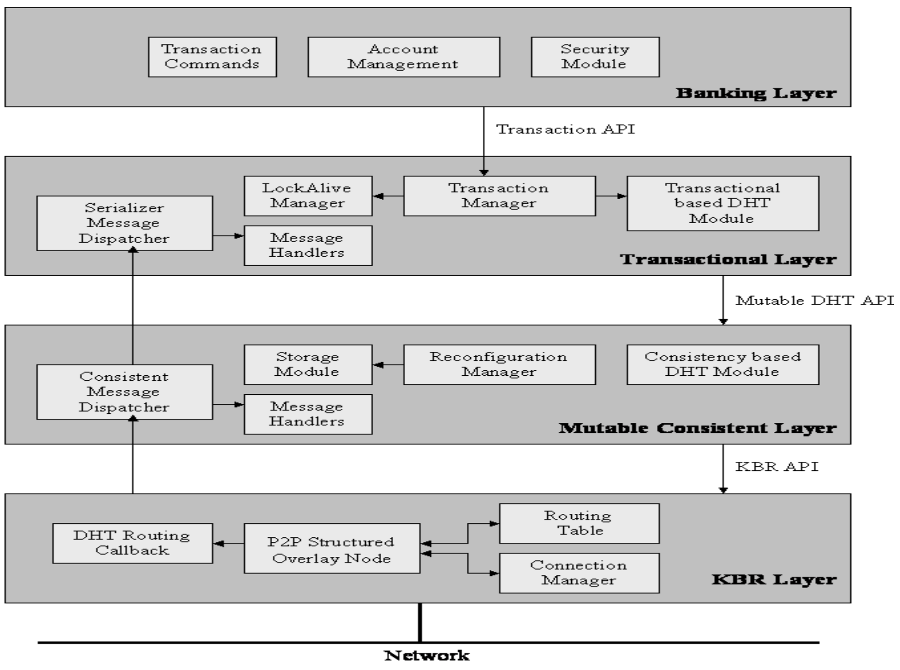


Figure 5 CMS Architecture

Technically CMS copes with a dynamic environment (nodes come and go) and where rate of account modifications is high (account is updated for each economic transaction decided in the market). From the economical point of view, its main purpose is to regulate and limit customer's consumption power to avoid overloading and provide an incentive for providers to share their resources. CMS is built with a layered architecture and wrapped as a single fractal component. It offers a single API and is (described in the Appendix B.5) used by customers to perform the necessary transfer of funds between accounts. The layered architecture and the main responsibilities of each layer are depicted in

Figure 5.

The report [SS-5.CMS-1] presents the detailed technical architecture and design of the CMS. This report is sent as a companion document with this deliverable.

5.4.4 Market factory

The specification of the Market factory and its implementation design will be available in September 2008.

5.4.5 Negotiator

Negotiators are software agents that trade on behalf of consumers and providers. Negotiator behaviours are expected to be dependent on the needs of the VO and the applications that execute in the VO. We focus on Negotiators to allocate computational resources for elastic applications, in particular bag-of-tasks applications such as the gMovie. Such applications tolerate variability in performance and hence adjust quantity of resources.

Virtual Organisations execute Negotiator agents encapsulating the negotiation process and offer brokering interfaces to resource managers. We have started implementing Negotiator agents based on assumptions described within the companion document [CAS-8]. This implementation and resulting evaluation will be reported in future deliverables.

5.5 Technological choices

This section presents the rationale for the adoption of the underlying technologies used in the design of the services and tools of the G4A resource market-place.

5.5.1 Component model and Fractal

The auction server is conceived to facilitate design of new auction formats and to facilitate developing repository services to select appropriate auction formats by assembly of required rules (of the auction). We are extending the design of the auction server to handle distributed auctioneers; for scalability when number of participants increase and for availability. DMIS and CMS are decentralized systems whose components need to be deployed on multiple nodes of a large scale distributed system.

Component technologies have proven their advantages: reuse, modularity, specialization, composition and reconfiguration. D2.2 describes implementations of the Fractal model including its Architecture Description Languages and run-time software to deploy applications on large scale systems. Deployment involves instantiation of application components and establishment of bindings between these.

Recent years have also seen the emergence of tools that bridge CBSE (component based software engineering) and Service Oriented Architectures. This trend will continue since CBSE is appropriate to design and develop *back-end* logic and SOA and Web Services is well suited to integrate, wrapping and exposing functionalities in a platform.

5.5.2 Distributed Component Management System

Aggressive management capabilities are required when decentralized and distributed services execute in harsh environments. This is the case with GRIMP services. The Currency Management System (CMS) is a peer-to-peer application and is able to self-organize and self-optimize the load through internal reorganization of stored items between nodes, when a node joins, leaves or fails; it lacks the ability to self-manage the system as a whole. Using component model and DCMS (Distributed Component Management System developed in WP1) allows us to define general managing policies to self-manage CMS as a whole; e.g., it allows us to define a minimum number of CMS running nodes without administration penalties; enables us to guarantee an optimum load for each node by adding or removing nodes from the system as the load against it changes over time.

We are extending the design of CAS to support multiple instances of the *Auction* component. Clearing algorithms are potentially hard problems, in particular in the case of combinatorial auctions. A distributed auction component may reduce the time to compute allocations.

The DCMS technology developed within Grid4All combines the best of component technologies, overlay technologies and feedback-control based autonomic management patterns.

5.5.3 Overlay and peer-to-peer technologies

The operational model that we envision is that consumers and providers create auctions on demand. This can also be done 3rd parties; value-added intelligent agents that monitor the market-place and instantiate auctions at the appropriate place and time. Participants select the auctions (amongst running ones) at which they trade; participants require suitable information from the markets to bid; to set prices, adjust times, deadlines and resource quantities. This watcher service is provided by the DMIS. DMIS should scale in number of messages and market participants. Key-based routing and DHT technologies offered by overlay services are promising to address this.

CMS should scale in the number of objects (accounts and its related transaction logs) that it stores and with the number of account transactions. DHTs (provided by the overlay network) are effective to store and retrieve large number of objects in a scalable manner. CMS enhances a specific DHT implementation (namely, the DKS P2P middleware) to improve storage guarantees and to decrease the delay when several objects need to be modified atomically.

5.6 Usage within Grid4All

This section recapitulates the two usage scenarios requiring allocation of resource leases:

- gMovie demonstrator: Section 4 has explained how gMovie management interacts with the Reservation Manager[14]. gMovie is an adaptable bag-of-tasks application that adjusts to varying quantities of compute nodes; higher the number of nodes, shorter the completion time. Willingness to pay is expressed directly by the user (who needs to transcode); quicker completion time may imply a higher cost. The user is expected to specify the earliest (cannot use the result before this time) and latest (result is useless after this) desired completion times. The objective is to execute the application within the required time span minimizing the cost of resources.
- Network simulation for classrooms: Network simulation lessons are organized in a school VO. Lessons run over two days. On the first day, students in groups prepare scripts, desired network topology and a range of parameters to simulate. Number of parameters differs from one group to another. Simulations should complete before the next day. The objective is to maximize the number of successful runs (subject to a maximum willingness to pay). Fairness between the student groups should be guaranteed; all groups should progress fairly.

5.7 Conclusions

Chapter 5 has presented the status regarding software prototypes[15] of the G4A resource marketplace. The first prototypes have been implemented and the different integration points have been designed. We focus integration (software prototypes) towards the usage described in section 5.6. A comprehensive user guide of the market-place tools will be provided at the 30th month.

[14] Explained in D2.2
[15] Currently, the software is available on demand. We will soon host all the software components on the common G4A gforge server.

5.7.1 Integration and prototype work

The following software integration is ongoing:

- Advertise markets at SIS and query for markets at the SIS: Usage of SIS APIs is fairly trivial since they are well documented and clear. The Negotiator agent (for bag-of-task applications) will use this API to query and select markets. Agents that initiate markets will advertise markets at the SIS.
- Publish dynamic market information using DMIS APIs: DMIS is a decentralized peer-to-peer service. The service will be used by (a) Market, to publish information and (b) Negotiators, to subscribe/query information. A DMIS peer executes on every node of the market-place. Auctions deployed on a node will publish their events to the local DMIS peer using the Market Application Layer provided by DMIS.
- Negotiation agent and Reservation Manager: We plan to implement a prototype of the Negotiation agent to satisfy needs of the usage scenario described previously.
- Integration of combinatorial auction within the Auction framework: The current code implements a linear integer formulation of the winner determination problem that decides the allocations. This requires solvers such as CPLEX. We aim to design the interfaces and support for exact optimization based solutions such that minor modifications to the model does not imply complete rewriting of the clearing component.

The Figure 6 illustrates the interactions between the different modules that have been previously described.

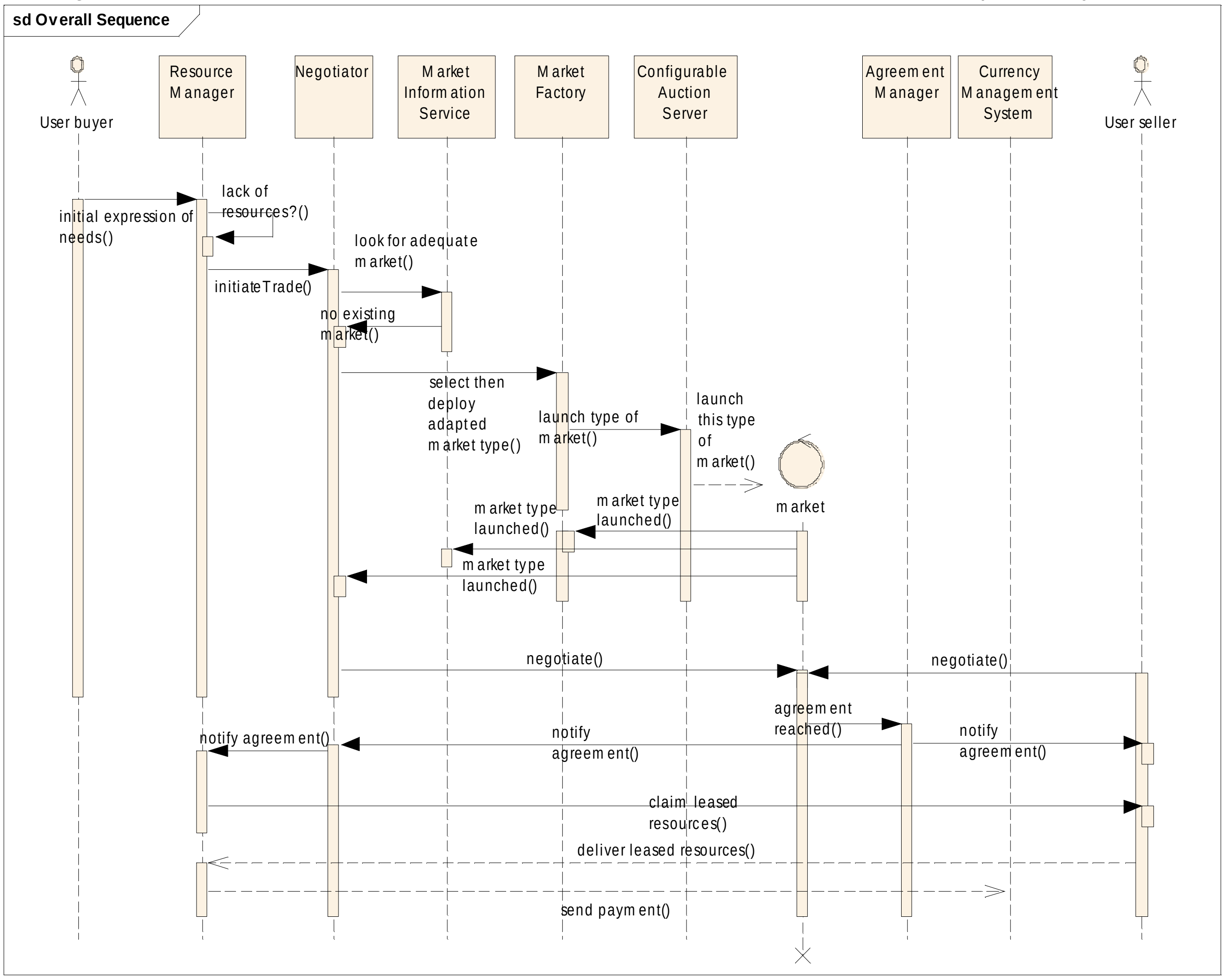


Figure 6 Interactions between GRIMP and SIS modules illustrating allocation of resources

5.7.2 Design

Ongoing design work includes:

- Pricing models for combinatorial auction: We are devising a pricing model to compute per-item (commodity) prices per-time slot for computational and storage resources.
- Distributed auctioneers: Currently an auction server has a single instance of the Auction or Market component. We plan to extend this to support multiple instances of Auction and Market component. Two reasons motivate this: ensure that the service is not completely lost on node failures; handle increase in load represented by number of registered participants and the number of bids.
- Self-management for Currency Management System: The CMS sub-system will use the DCMS framework for advanced self-management capabilities.

6 Semantic information service

Computational and storage resources are traded applying economic models. Consumers and suppliers negotiate at auctions initiated by resource providers, by resource consumers, or by third parties[16]. The market-place is populated by multiple, simultaneous and independently operating trading instances.

To support discovery of resource markets and of *services*, we propose the Semantic Information System (SIS) to publish and discover services. SIS is an information service where OWL-S service profiles are published. End-point references to these services are discovered by sending queries to it. SIS matches required service descriptions against offered services. The Grid4All resources ontology [SIS-9] is used to select markets. Queries are matched against services advertised as OWL-S profile specifications.

6.1 SIS Work flow description

SIS provides matching and selection services for peers that offer or request resources and services. Queries may be issued by software agents or human users to discover and select advertised markets and services. Queries for services are matched against their OWL-S profiles and results are ranked according to resources/services matching characteristics and providers'/consumers' features.

As shown in Figure 7, SIS has three main modules; to process advertisements, to match queries to advertisements and to select (rank) matches. The Ontology registry is used to store domain ontology and facts. Queries to discover generic services are matched against published OWL-S service profiles and queries to discover markets are matched against the published offers and requests.

SIS exploits the Grid4All resources ontology [SIS-9] to discover markets and the OWL-S services profile specifications to discover services. Other types of market related, application-oriented and offers/requests related properties can also be exploited for matchmaking. Entities (human or software) pose queries to SIS. The matchmaking component matches queries with entries in the registry and ranks results according to their similarity as well as the providers/buyers features. Queries may be requests (seeking matching offers) or offers (seeking matching requests). The *Request* concept describes the resource needs of a consumer. The *Offer* concept describes the resources offered by a provider. *Request* and *Offer* are sub-concepts of *Order. Requests* and *Offers* are *traded* at *Markets*.

The following sections present the details of the main sub-systems.

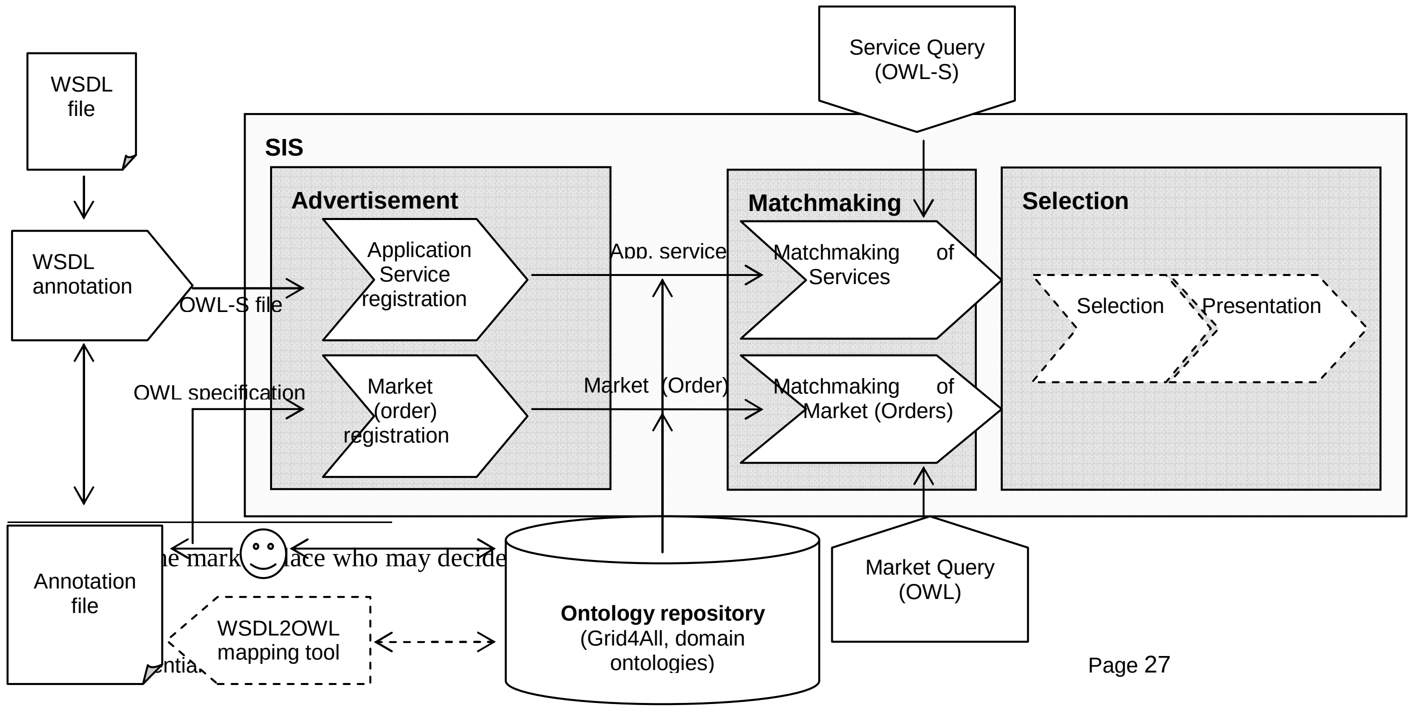


Figure 7 SIS Workflow

6.1.1 Resource market discovery

Advertisement and discovery of markets are based on the concepts of *Order*, *Request* and *Offer*. A *Request* describes the resource properties, the quantity, time intervals of their allocation and the price the consumer is willing to pay. An *Offer* specifies the exact resources (quantity of offered resources) that a provider trades in specific time intervals and price: This is in contrast to the specification of requested resources, where the consumers may request a class of resources. Both *Request* and *Offer* are sub-types of *Order*.

We deal with two types of grid resources: computational and storage resources. *Tradable* resources may be either Compute Nodes or Clusters.

- A Compute Node is a type of Composite Resource that comprises exactly one Computational Resource and any number of Storage Resources.
- A Cluster is an Aggregated Resource comprising a set of Compute Nodes.

Orders may be “atomic”, representing a single resource, or "complex", representing a bundle of more than one type of resource. Multiple orders of the same type (offers or requests) may be connected using an XOR (exclusive OR) (respectively, AND) connective operators.

Auction-based markets are created to trade resources by consumers who have *Requests* (to buy resources) or by providers who have *Offers* (to sell resources). 3rd parties may create markets trading *Orders*. In this final type, multiple sellers and multiple buyers may participate. SIS provides a registry of the published e-markets, together with a retrieval and ranking service for those markets: Participants query the SIS submitting orders (i.e. requests and offers). These are matched against advertised orders of the opposite type. Query results are ranked according to the preferences and intentions of providers and consumers, as well as according to the characteristics of resources and markets. Main features related to market discovery are:

- Publishing or advertising markets, by providing market-related requests or offers, as well as information about traded resources and services, and
- Querying in order to obtain a list of relevant markets according to the resource/service ordered (as a consumer request or as a seller offer) as well as market characteristics.

6.1.2 Advertisement

This functionality allows insertion of *offers* and *requests* in the SIS registry. *Orders* (offers and requests) contain information about the entities that are traded at their associated markets, that is, resources, information about the related markets, the participants i.e. providers and prospective consumers of resources and services, as well as attributes of the orders themselves. Such descriptions are instances of the Grid4All ontology schema and are stored in SIS in OWL format.

Markets may be advertised by API (for software agents) or using a web-based user interface (for humans). No authoring of formal descriptions of input information is required from users to create and submit an ontology instance (SIS registration). Consumers and providers of grid resources have to subscribe to the SIS in order advertise initiated and running market services. Specific APIs are available for consumers that advertise requests and providers who advertise offers.

- ➢ Providers advertise *forward* markets that trade their *Offer objects*. Offers may also be bundled, that is as a list of XOR or substitute offerings.
- ➢ Consumers advertise *reverse* markets that trade their *Request* objects. Requests describe the resource types, their quantities and characteristics.

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

Orders are generalizations of Offers and Requests. Orders contain the following information.

- Description of the market where the resource/service is to be traded: location of the market, starting and closing time of the market.
- Description of the technical characteristics of the traded service or resource in terms of capacity, quality of service, time of availability, etc.
- Description of pricing policy, initial price auction price (minimum price for a forward auction and maximum for a reverse auction).
- Information about the provider or consumer.

The actors, inputs, outputs, pre-conditions and post-conditions (effects) of supported use cases for market query are briefly presented in table Tableau 1

Advertised by	**Provider**	**Consumer**	**3rd party**
Actor	A provider who initiates a forward market.	A consumer who initiates a reverse market.	Any agent initiating an exchange or double auction.
Input	Offer and market properties	Request and market properties	Abstract Order and market properties
Output	N/A	N/A	N/A
Pre-condition	Provider registered at SIS	Consumer registered at SIS	Initiator registered in SIS
Post-condition	Advertisement is stored in SIS registry	Advertisement is stored in SIS registry	Advertisement stored in SIS registry

Tableau 1

Examples of advertisements and API to advertise markets are described in Appendix A.

6.1.3 Market Querying

The Query interface returns a ranked list of advertised markets. Queries filter advertisements based on the *Order* and some *Market* properties. Advertised markets are matched against the query filter and ranked according to selection criteria. Semantic descriptions of advertised markets that fulfil the query criteria are identified through type-based matching. Matched results are ordered by the selection mechanism based on characteristics including the capacity of resources, preferences and intentions of providers and consumers. The ranking process provides an ordering of results reflecting the user preferences (e.g. preference on specific peers) performed by the selection component of the SIS. The returned results may be: a list of resources/services, the list of the corresponding markets or may also be the list of providers or consumers. This is chosen by the user performing the query. **The SIS API that clients use (either developers who use programmatic API or human agents who use web-based interfaces), do not require knowledge of ontology specific query language.** Query is performed by providers and consumers. The table [Tableau 1] gives the inputs, outputs, preconditions and post-conditions (effects) of supported use cases for market query.

Appendix A provides examples of queries and the API methods to query the SIS for the purpose of discovering markets.

Queried by	**Provider**	**Consumer**
Actor	A provider who wants to discover a market.	A consumer who wants to discover a market.
Input	Offer and market properties	Request and market properties

Output	Ordered list of end-point references to markets or a list of consumers.	Ordered list of end-point references to markets or a list of provides.
Pre-condition	Provider registered at SIS	Consumer registered at SIS
Post-condition	Query is stored in SIS registry	Query is stored in SIS registry

Tableau 2

6.1.4 Service discovery

SIS may be used to discover services. A service provider registers (*advertises*) a service in the SIS in order to be *discovered* by prospective clients of this service (service consumers).

6.1.5 Service advertisement

Providers submit a service description in WSDL, along with annotations in a document named External Annotation File (EAF). The EAF describes the mapping between advertised service I/O types and concepts in OWL ontology previously stored in the SIS registry. The SIS automatically generates and inserts the corresponding *OWL-S profile* specification in its registry. Advertisers may prepare the annotation document by using the annotation tool described in section 6.2. An important part of the registration process is validation. Before registering, the provided information is inspected to ensure that there is no type mismatch and that the consistency of the knowledge base that stores registered descriptions is maintained.

Programming API to prepare and send advertisements are described in Appendix A.

6.2 WSDL Annotation

WSDL annotation is an important part of matchmaking and selection. It provides mappings between WSDL I/O types and the corresponding domain ontology. Annotation can be performed interactively or automatically if WSDL parts requiring annotations are satisfactorily described. In interactive mode, humans provide mappings between WSDL I/O parts and ontology classes (this process is called semantic annotation). We have devised a mechanism that computes mappings between WSDL I/O parts and OWL classes to automate this process. Humans, in general developers have to provide descriptions and comments concerning the intended meaning/use of these I/O parts and types used (in contrast to semantic annotation, we call this process annotation). Annotations may also be fetched from code documentation**.**

6.2.1 Introduction

Lack of explicit semantics in WSDL specifications reduces effectiveness of discovery. We annotate a WSDL service description using an External Annotation File (EAF) that stores semantic annotations of WSDL I/O parts. The annotation file is separated from the WSDL file to track changes and to separate concerns of the developer from those of the annotators.

We have developed the WSDL-AT, WSDL Annotation Tool for human annotators to support:

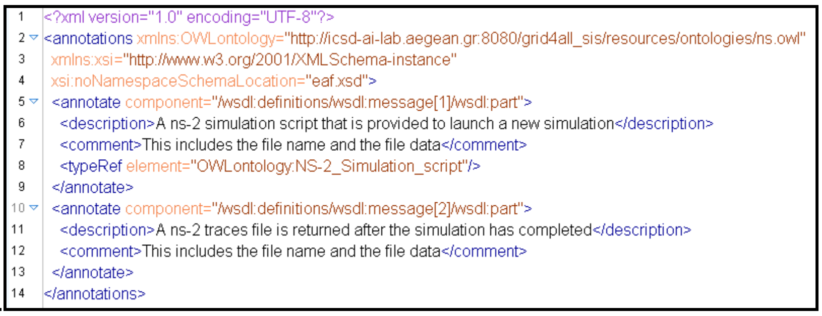
- ➢ Manual annotation of WSDL elements with natural language descriptions.
- ➢ Automatic semantic annotation of WSDL elements that refers to classes of domain ontology.
- ➢ Validation of the generated semantic annotations.

Details concerning the functionality of this tool are provided in the Appendix A.

6.2.2 External Annotation File (EAF)

EAF is XML encoded and based on an XML schema. The EAF file provides “slots” for the (semantic) annotation of WSDL elements. The XML schema specifies elements for "comments", "description", as well as "type reference" to ontology classes, for each of the WSDL elements. The annotation file uses the <typeRef> elements to map each WSDL part element to an ontology class, and hence to semantically annotate the WSDL element.

The EAF annotations are aligned with the WSDL specifications via XPATH expressions. Comments and descriptions are some of the possible types of textual (or other media) information and can be extended to other textual media. An example of EAF file is depicted in the following snippet.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<annotations xmlns:OWLontology="http://icsd-ai-lab.aegean.gr:8080/grid4all_sis/resources/ontologies/ns.owl"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="eaf.xsd">
 <annotate component="/wsdl:definitions/wsdl:message[1]/wsdl:part">
  <description>A ns-2 simulation script that is provided to launch a new simulation</description>
  <comment>This includes the file name and the file data</comment>
  <typeRef element="OWLontology:NS-2_Simulation_script"/>
 </annotate>
 <annotate component="/wsdl:definitions/wsdl:message[2]/wsdl:part">
  <description>A ns-2 traces file is returned after the simulation has completed</description>
  <comment>This includes the file name and the file data</comment>
 </annotate>
</annotations>
```

Figure 8 XML instance of EAF XML Schema

6.2.3 WSDL-AT Functionality

The WSDL-AT (WSDL Annotation Tool for human annotators) facilitates the human annotator in the following tasks:

- Creation and authoring of EAF for WSDL files: Users create a new EAF when starting an annotation process. If however an EAF already exists, the WSDL-AT imports and depicts the existing annotations and provides editing facilities to the user. During the annotation procedure, WSDL-AT provides a set of ontology, from which the most appropriate ones may be chosen to perform the WSDL-to-OWL mapping process. This can also be used a reference to validate the semantic annotation of WSDL elements.
- Mappings of WSDL I/O parts to the classes of a given ontology: WSDL-AT initiates the WSDL-to-OWL mapping process. When human-created annotations (provided in natural language within the "descriptions" and "comments" elements) of WSDL part elements are entered, WSDL-AT automatically computes and suggests OWL classes (from the domain ontology) that match the WSDL I/O part elements (i.e. the semantic annotations of WSDL part elements). This automatic computation is performed using algorithms of the ontology alignment paradigm. Suggestions for each WSDL part element are returned as a ranked list; the higher position in this list implies a stronger matching proposition. Human annotators can inspect the domain ontology and the suggested mappings produced by WSDL-AT and may change the semantic annotation of a WSDL part element by selecting an alternative ontology class (from the domain ontology hierarchy).

WSDL-AT is a platform-independent stand-alone application, with a graphical user interface (GUI). For more information and a use case example, please refer to the Appendix A.

6.2.4 Service discovery through querying of SIS

Clients submit queries to discover services advertised in the SIS registry. A query is specified by a list of input and output types of the required service. These types are OWL classes defined in domain ontology stored in the SIS. Matchmaking consists of discovering semantic (similarity) relationship between the I/O parameter types of registered services and the I/O types specified in the query. Advertised services whose I/O types match are considered as satisfying the query. Reasoning services identify exact as well as partial matches. Matched services are ranked by the selection component of the SIS that uses the preferences of service consumers and providers. The output of a service query is a ranked list of endpoint references of services which match query criteria.

6.3 Main software modules

SIS consists of the *matchmaking* and the *selection* component. These two components interact through an internal API and are together accessible though a uniform external API. The API provided by the SIS is available as a set of web services to:

- Provide access to system features to agents such as consumers and providers,
- Facilitate interoperability with other Grid4All components.

The Appendix A presents the SIS API to perform service advertisement and querying.

6.3.1 Matchmaking module

Matchmaking functionality of resource markets and services are presented in the two subsequent sections.

Resource Market Matchmaking

The Grid4All ontology has been engineered so that retrieval of information about markets, orders, and traded resources proceed by means of a) the automatic classification of individuals by computing their inferred types and b) SPARQL queries to filter matching individuals by market-related properties and orders' constraints. To leverage the classification mechanism, and according to the requirements for offers and requests, we have represented orders in the following way: Resources *offers* are represented as individuals of class Offer and *requests* are represented as defined subclasses of the class Request. Hence, matching offers are classified under specific request subclasses. Individual markets are classified under their corresponding subclasses of the Market class (e.g. depending on whether they are consumer or provider-initiated markets). Only those markets whose Request/Offer match the client's resource specification, the prices, lease specifications and the market properties are retrieved. For example, even if the resource attributes match, if the price or the number of time-slots requested do not match those of the offer, then no matching occurs.

An example of the functioning of market matchmaking is provided in Appendix A.

Service Matching

OWL-S service profile descriptions generated by automatic translation of WSDL specifications are stored as advertisements. A list of input/output types is submitted as queries to the SIS. We assume that the I/O types in a submitted OWL-S profile document are already known, i.e. they refer 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 will be an input parameter and an output parameter of the respective types.

The matching of advertisements to a submitted query is divided in two main stages: a) matching of inputs, and b) matching of outputs. For the matching of inputs and outputs, the direction of the subsumption relation is important for (a) the input types to ensure proper execution of the service and for (b) the output types to fulfil the demands of the service requester ([SIS-1]).

Three basic types of matching are defined in the context of Grid4All services: Exact match, "Subsumes" match, and fail ([SIS-2]). 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 $\Leftrightarrow \forall IN_S \exists IN_R: INS \doteq IN_R \wedge \forall OUT_R \exists OUT_S: OUT_R \doteq OUT_S$. For every input type of the advertised service one equivalent input type of the required service is found. Also, for each output type of the required service one equivalent output type of the advertised service is found. The service I/O signature perfectly matches with the request with respect to their formal semantics.

- "Subsumes" match. Request R subsumes service S $\Leftrightarrow \forall IN_S \exists IN_R: IN_R \sqsubseteq IN_S \wedge \forall OUT_R \exists OUT_S: OUT_S \sqsubseteq OUT_R$. For each input type of the advertised service exactly one input type of the required service has been found, which is at least subsumed by the input type of the advertised service. This means that the advertised service might be invoked with a more specific input than expected. The output types of the required service subsume the output types of the advertised service or are equivalent to them. This means that the required service might receive a more specific output type than expected. Additionally, for all output types of the required service a successfully matching counterpart of the advertised service is identified.
- Fail. Service S fails to match with request R in any of the ways described above. This means that one of the following holds: a) at least one input type of the advertised service has not been successfully matched with one input type of the advertised service, and so the service cannot be executed properly, or b) at least one output of the required service has not successfully been matched with an input of the advertised service.

For more information and a use case example, please refer to the Appendix A.

Selection module

The Selection Service (SS) is in an internal module of the Semantic Information System (SIS) and is used to rank matching services and markets. SS narrows down and ranks the providers list found by the Matchmaking Service (MS). To narrow down (or only to rank) the list of providers, MS invokes the selectProviders method of SS by passing it: the query identifier and type, the query source identifier (i.e. the consumer), the set of providers that can deal with the query, and the size of the providers list that the consumer is waiting for. Then, SS module ranks (or narrows down) the list of providers according to (i) the preferences that consumers have towards providers (regarding providers' reputation for example), (ii) the preferences that providers have towards queries (regarding which data or service is concerned by the query), and (iii) the query load of providers. With this aim, we assume that consumers and providers declare at any time their preferences to SIS so as to get those providers and queries they prefer, respectively, at the top of (or included in) the providers list returned by SIS. To do so, consumers invoke the "setConsumerPreferences" method and providers invoke the "setProvidersPreferences" method to set their preferences at SIS. It is worth noting that, conversely to consumers, providers should invoke the "getQueryTypes" method to discover the types that a query can be. Some examples of these types are given in previous section. Moreover, a consumer and a provider can also define default preferences for those providers and queries, respectively, it does not know (see the Annex for details about the SS's API).

A natural way to rank providers is considering a consumer-centric fashion, as several e-commerce applications do. This generally takes into account the consumers' preferences (denoted by vector CI). This may however penalize providers' preferences (denoted by vector PI)). Similarly, if the ranking service considers only the providers' preferences when allocating queries, consumers may quit the mediator by dissatisfaction, which in turn may cause the departure of providers. We hence balance consumers' and providers' preferences to satisfy both. Given a query q, a provider p is scored by considering both its preference for performing q and the preference of consumer c (who issued q) for allocating q to p. That is, the score of p regarding query q is defined as the balance between the c's and p's preferences as follows:

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

The parameter ***w*** ensures such a balance and takes its values in the interval of [0...1]. To guarantee equity at all levels, such a balance should be done in accordance to the consumer and providers' satisfaction so that the less satisfied one be paid more attention. Overall, SS aims at equally satisfying buyers and sellers so that they almost have the same chances of doing business and getting interesting resources or services in the long-run. Satisfaction, in our context means, how well preferences are met by queries a seller gets and by resources/services a buyer gets. To this end, MS informs SS of buyers' final choices and of queries allocated to sellers: MS does so by invoking the "informFinalSelection" method of SS by giving the query source identifier and the set of selected sellers as parameter. Details and validation can be obtained from [SIS-6].

6.4 Design and Implementation

Figure 9 depicts the relations among the various SIS components and the functions which are available to the users.

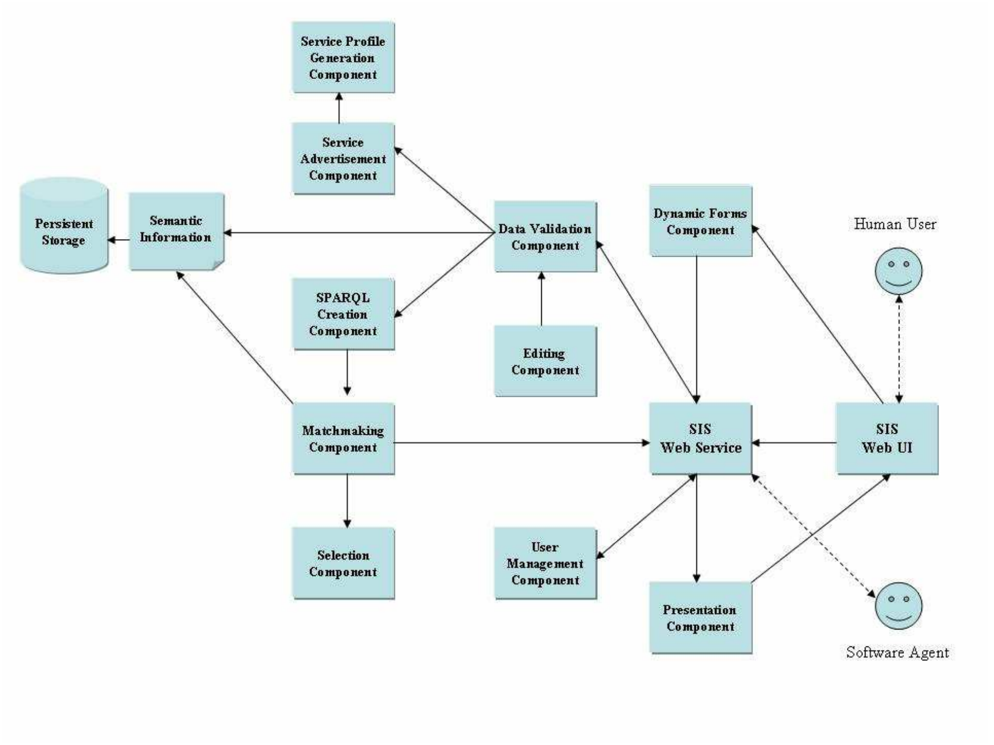


Figure 9 SIS Architecture

6.4.1 Technological choice

SIS is implemented using Java technologies (Java Server Pages (JSPs) and Servlets) and provides Web-form-based interface (HTML forms and Javascript functions) for human-users to advertise their resources or services or submit queries. The Jena framework [SIS-3] is used to represent semantic information (e.g. representation of resources and markets, semantic matchmaking) and the Pellet [SIS-4] for reasoning.

Jena provides a rich API to manage ontology. In Jena terminology, a knowledge base is called a “model”. The basic units in a Jena model are resources and statements (the concepts originated from RDF). The concept of resource is fairly complex, since almost every entity (Classes, individuals, properties, and even statements themselves) can be regarded as resource. The Jena API provides classes and interfaces for all the concepts of RDF(S) and OWL languages: Other than classes and properties, such concepts include subclass relations, property restrictions, RDF data types, etc. In addition, Jena uses a simple SPARQL engine, ARQ, through which SPARQL queries can be executed. Finally, Jena offers the capability of attaching inference engines, such as Pellet, to the models. When an inference engine is attached to a model, query processing can be enhanced because inferred statements, which may provide answers to queries, are discovered. Such an enhancement, of course, comes at the cost of query processing time.

The SIS uses a MySQL database (to store RDF/OWL models) to store the ontology. When a user makes an advertisement or query the SIS stores the relevant semantic descriptions in the database and, if necessary, reasons with the updated ontology in order to obtain inferred statements related to the new registered ontology entries. Inferred knowledge is cached for future use. This means that automatic classification, which is a vital part of the matchmaking mechanism for grid resources, is executed exactly after object registrations, and not during query processing. Caching improves responsiveness of query processing.

The implemented system has been tested in the Apache Tomcat server. In general, the loading times during advertisements were acceptable, but the system has not yet been load-tested. Also, the SIS portal requires relatively large amounts of Java heap space, due to the inferred statements which are cached after object registrations. This shortcoming may either be resolved by the persistent storage of inferred statements in the database, or by pruning the cache, i.e. removing statements that concern past offers, requests, markets, etc.

The AXIS 1.4 Web services framework [SIS-7] is used for enabling web service access for the SIS API. The SIS API, supported by AXIS, conforms to the WSDL 1.1 specification. Thus, the overloading of operations of the API, such as queryMarket, is supported.

6.4.2 Conclusions

The implemented SIS offers basic functions for human users and software agents to advertise semantic descriptions of resources (specifically, of markets trading resources) and services, and submit queries and obtain results using the matchmaking mechanisms described earlier. Human users interact with the SIS through a portal. Software agents access it using Web Services interfaces exposed by the SIS. Web Service interfaces also facilitate automation of testing and benchmarking.

Currently, the results obtained by the matchmaking process are only distinguished as either being exact matches of the submitted queries or “subsumed” matches. Ranking of the retrieved results could be more fine-grained, and involve information related to ontology structure and to preferences of individual users. [SIS-2], describes additional matching types suiting the first requirement. [SIS-6] considers users’ intentions and preferences regarding other users, either providers or consumers. The selection component of the SIS, which has been designed by a separate group within the Grid4All project, takes advantage of the user preferences in order to rank matched results. Future work on ranking involves using the structural information of the ontologies used in the matchmaking, e.g. the semantic distance between subsuming and subsumed concepts.

Future work also includes testing of scalability to ensure that the SIS will be able to support a large number of users (either offering or requesting resources). The current implementation of SIS is centralized. It is our intention, therefore, to design a decentralized version of the SIS, which will deal with the issues of scalability and availability.

7 How they all fit together

This section gives an overview of how the major functionalities described in this document and those described in D2.2 work together.

> *End users and members of Virtual Organizations should be able to execute their applications without needing to decide where (on which physical machines) and when. Application should continue execution with minimal or no manual intervention even if compute nodes were to fail or leave. VOs should have means to allocate computational resources and execute applications on these resources. Users and administrators should be able to provide hints on required qualities of service*[17].

This is a digest of a set of related requirements that have been presented in D4.7. We have selected the *gMovie* application to demonstrate this. This application converts video films from one compression format to another and belongs to a category that has been called embarrassingly parallel. It may be implemented as bag-of-tasks application by dividing the entire film to be processed into smaller chunks that are processed independently. Applications of this category are adaptive and scale as number of processors is increased (up to a threshold). The Collaborative Network Simulator Environment application described within D4.3 also belongs to this category. The application will execute within a Virtual Organization whose pool of resources may be resized by leasing compute node at resource markets.

The main parameter of interest to the user is the desired completion time (or deadline). Earlier completion time may imply a higher cost. Hence the second related parameter is the user budget value. These two user indications drive resource allocation decisions; system leases resources that are necessary to complete execution within the desired deadline, subject to the available budget.

gMovie application is adaptive, i.e., adjusts to variable parallel processing (computational) capacity. The first prototype of the demonstrator will exercise the auction server implementing the K-Double Auction mechanism, to allocate leases of compute nodes represented as a composite indivisible resource (computational and storage). In future work, we will extend this to use the combinatorial auction mechanism.

This demonstrator will exercise the following modules:

- Scheduling service: It is used as an offline planner. Based on estimate of CPU (quantity, quality, duration) that can be allocated with the given budget, the scheduler gives an estimation of completion time. Multiple iterations may be required with the scheduler.
- Reservation manager: It uses the Negotiator to find estimates of resource configurations (quantity, quality and duration of computational capacity), given a budget value. Once user has confirmed the budget (based on acceptable completion time), the RM requests the Negotiator to lease resources.
- Negotiator: This is a central module that uses all the services of the market-place. It queries DMIS to obtain estimates of resource configuration (quantity, quality, duration) for a given budget value. It queries the SIS to select from currently operating auctions. It registers and participates (submits prepared bids) at a selected[18] Auction. If successful in negotiation it receives *Agreements* from the Auction. The negotiator uses the provider's end-point reference available in the Agreement object to obtain the end-point addresses of concrete resources corresponding to the allocated leases.
- Deployment service is used to deploy XtremWeb slaves on allocated nodes by application manager on notification of resource availability.
- Execution of gMovie tasks are managed by the XtremWeb desktop computing middleware.

[17] The exact parameters defining QoS are clearly dependent on the application itself.
[18] Strategic negotiators who may simultaneously participate in more than one auction for the purpose of acquiring the same set of resources is out of scope.

7.1 Limitations and restrictions

- Negotiation agents: We will focus on developing negotiator agents to satisfy the purposes of the demonstrator. Complete buyer agent and seller agent frameworks are out-of-scope within our project.
- Provider side management: Designing provider-side resource managers are out of scope. We will implement proof-of-concept provider side management for purposes of the demonstrator.
- Payment: The demonstrator will not be integrated to use the services of the currency management system to transact payments.
- Self-managing execution management: The execution management of task farms will not use the DCMS in its design and implementation.
- Multiple applications: Each application is expected to manage its own budget. Hence each application runs within its own set of resources. There is no global VO-wide budget and resource management.

8 Conclusions and Future work

This document has presented the current implementation status of scheduling service, the market-place services (CAS, DMIS, and CMS) and the SIS. Usage and integration of these software modules will be demonstrated through an application as described within section 7.

We have identified the main future work concerning each software module. It is clear that a single demonstrator does not evaluate a dynamic market-place with multiple contending applications. A separate document [CAS-8] presents a detailed evaluation plan with a goal to gathering insights on the feasibility of using market-based methods to allocate resources for Democratic Grids. A second key future work will investigate how the Semantic Information Service may be distributed or decentralized.

An important aspect of democratic grids is that of management. WP1 has released the DCMS which permits developing of self-managing applications. Due to time and resource constraints, the execution management does not leverage this middleware. We plan to rectify this by providing the design of master-slave behavioural skeleton using the DCMS. Future work will address how DCMS and core VO services could be used to develop self-managing market-place services.

9 References

SS-1. Towards Soft Real-Time Applications on Enterprise Desktop Grids -- Derrick Kondo , Bruno Kindarji, Gilles Fedak et Franck Cappello -- In Proceedings of 6th Innternational Symposium on Cluster Computing and the Grid CCGRID'06 Singapore, 2006

SS-2. Characterizing Result Errors in Internet Desktop Grids D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello -- European Conference on Parallel and Distributed Computing -- EuroParRennes, August 2007 -- Best paper award

SS-3. Towards Efficient Data Distribution on Computational Desktop Grids with Bittorrent -- Baohua Wei, Gilles Fedak et Franck Cappello -- Future Generation Computer Science FGCS -- Selected paper from ISPDC'05, 2007

SS-4. BitDew: A Programmable Environment for Large-Scale Data Management and Distribution -- Gilles Fedak, Haiwu He and Franck Cappello --INRIA Research Report 6427, January 2008

SS-5. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet -- Freund, R.F.; Gherrity, M.; Ambrosius, S.; Campbell, M.; Halderman, M.; Hensgen, D.; Keith, E.; Kidd, T.; Kussow, M.; Lima, J.D.; Mirabile, F.; Moore, L.; Rust, B.; Siegel, H.J. -- Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh Volume , Issue , 30 Mar 1998 Page(s):184 – 199

MIS-1. M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. Technical report, Stanford University, 2003.

MIS-2. F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common api for structured peer-to-peer overlays. 2003.

MIS-3. Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

MIS-4. Peter Pietzuch, David Eyers, Samuel Kounev, and Brian Shand. Towards a common api for publish/subscribe. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 152–157, New York, NY, USA, 2007. ACM.

MIS-5. Rowstron,A. and Druschel,P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),* 2001, 329-350

MIS-6. René Brunner, Felix Freitag and Leandro Navarro, Towards development of Decentralized Market Information System: Requirements and Architecture; Parallel & Distributed Computing in Finance (PDCoF'08). In proceedings of the 22nd IPDPS, Miami, FL, USA, 2008

MIS-7. René Brunner, Felix Freitag and Leandro Navarro, On Efficient Routing Structures for Information Acquisition in Distributed Markets, submitted for publication

CMS-1. Leon, X. and Navarro, L. Currency Management System: a distributed banking service for the Grid. Technical Report n° UPC-DAC-RR-XCSD-2007- 6. Universitat Politètecnica de Catalunya, 2007

CAS-1. P. R. Wurman, W. E. Walsh, and M. P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27, 1998.

CAS-2. Vilajosana, X.; Marques, J.; Krishnaswamy, R.; Juan, A.; Amara, N.; Navarro, L. (2008): "Bidding support for computational resources". In Proceedings of the Second International Conference on Complex, Intelligent and Software Intensive Systems. Barcelona, Spain, March 4-7. p309 - 315.ISBN: 0-7695-3109-1.

CAS-3. http://www.sun.com/service/sungrid/index.jsp

CAS-4. aws.amazon.com

CAS-5. L'Ecuyer, P., Meliani, L., Vaucher, J. 2002. SSJ/ A framework for stochastic simulation in Java. In Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers (San Diego, California, December 08 – 11, 2002). Winter Simulation Conference, 234-242

CAS-6. http://www-sop.inria.fr/oasis/Vercors

CAS-7. Combinatorial Auction model for Democratic Grids, FTRD internal report

CAS-8. Negotiation agents and evaluation of Grid4All market-place, Grid4All working document

SIS-1. M. C. Jaeger, G. Rojec-Goldmann, G. Muhl, C. Liebetruth, and K. Geihs, “Ranked Matching for Service Descriptions using OWL-S”, Proceedings of KiVS, p. 91-102 2005.

SIS-2. M. Klusch, B. Fries, M. Khalid and K. Sycara, “OWLS-MX: Hybrid Semantic Web Service Retrieval”, Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, AAAI Press, Technical Report FS-05-01.

SIS-3. B. McBride, “Jena: Implementing the RDF Model and Syntax Specification”, Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001, pp. 74-83, 2001.

SIS-4. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur and Y. Katz, “Pellet: A Practical OWL-DL Reasoner”, Web Semantics: Science, Services and Agents on the World Wide Web archive, volume 5 , issue 2 (June 2007) pp. 51-53, 2007.

SIS-5. A. G. Valarakos, V. Spiliopoulos, K. Kotis, G. A. Vouros, “AUTOMS-F: A Java Framework for Synthesizing Ontology Mapping Methods”, I-KNOW 2007 Special Track on Knowledge Organization and Semantic Technologies, 2007.

SIS-6. J. –A. Quiane-Ruiz, P. Lamarre, P. Valduriez, “SQLB: A Query Allocation Framework for Autonomous Consumers and Providers”, Proceedings of the 33rd international conference on Very large data bases, pp. 974-985, 2007.

SIS-7. AXIS Web Services Framerowk, http://ws.apache.org/axis/, accessed June 9, 2008.

SIS-8. R. Krishnaswamy et al., “State of the art analysis and Requirements for the Grid4All Semantic Information” System, D2.1 deliverable, 2007.

SIS-9. K. Kotis et al. “The Grid4All ontology for the retrieval of traded resources in a market-oriented Grid”. WGISD08/CISIS 2008 proceedings, IEEE Computer Society, 2008.

A. Semantic Information Service

A.1. Market advertisement

A.1.1. Market advertisement interface

Market services are advertised by both providers and consumers. Providers advertise provider-initiated markets and consumers advertise consumer-initiated markets. SIS API defines the following operations for advertisement:

Method advertiseOffer

This method advertises an offer for cluster resource. This is specified by a provider in a forward market. The agent must register with the SIS before advertising. Method signature is:

```
String advertiseOffer(String providerId, ClusterOffer clusterOfferDescription,
                      String marketURL) throws InvalidURLException,
                                               InvalidAgentRoleException,
                                               InvalidDescriptionException,
                                               NoSuchAgentException
```

ProviderId is a unique identifier for the 'provider' role of the agent.

ClusterOfferDescription is a description of an offer as an instance of class `ClusterOffer`. **ClusterOffer** class is an object-oriented wrapper describing an offer stored in the SIS. The detailed specification of `ClusterOffer` is described in SIS API documentation. An *Offer* object contain information about the resources offered and about the market trading the offer, as described in the SIS Ontology.

`MarketURL` is the URL of the market service in which the specific offer is negotiated.

The method returns an identifier of the advertised order. It throws the following exceptions:

`InvalidDescriptionException` is thrown when the semantic information is malformed.

`InvalidURLException` is thrown when the market URL is null or malformed.

`NoSuchAgentException` is thrown when an invalid agent id is provided

`InvalidAgentRoleException` is thrown when a consumer id is provided

Other forms of the `advertiseOffer` method are available, through which other types of tradable resources are advertised. These forms have a second argument of the following type:

`ComplexClusterOffer`. Complex cluster offers contain multiple `ClusterOffer` descriptions, connected by one of the AND/OR/XOR operators. The AND operator implies that the advertising provider agrees to negotiate offered markets as a unique bundle that cannot be disaggregated.

`ComputeNodeOffer`. This class contains information about provider-initiated markets offering compute node offers.

`ComplexComputeNodeOffer`. This class contains information about provider-initiated markets offering complex computing node offers, that is, offers about compute nodes connected with AND/OR/XOR operators.

Method advertiseRequest

This method advertises consumer initiated reverse markets. Consumers advertise these markets along with their Request that describes the resources they require. They also provide information about the markets. Both kinds of information are stored in the SIS ontology after request advertisement.

```
String advertiseRequest(String ConsumerId, ClusterRequest ClusterReqDescription,
                        String MarketURL) throws InvalidURLException,
                                                 InvalidAgentRoleException,
                                                 InvalidDescriptionException,
                                                 NoSuchAgentException
```

This method registers a request for clusters, specified in a consumer initiated market.

ConsumerId is a unique id for the 'consumer' role of the agent.

ClusterRequestDescription is a description of a request as an instance of class ClusterRequest. Class ClusterRequest is an object-oriented wrapper for information about the intended characteristics of clusters which a customer would like to purchase in a reverse auction.

MarketURL - The URL of the market service that trades the Request.

Method advertiseRequest returns an identifier of the advertised order. It throws the following exceptions:

InvalidDescriptionException is thrown if the semantic information contained in parameter ClusterRequestDescription is malformed.

InvalidURLException is thrown if either service or market URLs are different than null but are either malformed or not available.

NoSuchAgentException is thrown if an invalid agent identifier is provided.

advertiseRequest method has another form for the advertisement of consumer-initiated markets for trading compute nodes. In this form, the method takes as a second argument an object of class ComputeNodeRequest instead of ClusterRequest.

Method advertiseOrder

This method is called by an agent, provider or consumer, to advertise a third-party initiated market, that is, a market that does not negotiate any tradable resources at the time of its initiation and advertisement. Markets of these types are discoverable through queries by both providers and consumers. The query should match the description of a particular order. The form of the method is the following:

```
String advertiseRequest(String AgentId, Order orderDesc, String marketURL)
                        throws InvalidURLException, InvalidAgentRoleException,
                               InvalidDescriptionException,
                                NoSuchAgentException
```

AgentId is a unique identifier for the 'consumer' role of the agent.

ClusterRequestDescription - A description of a request as an instance of class ClusterRequest. Class ClusterRequest is an object-oriented wrapper for information about the intended characteristics of clusters which a customer would like to purchase in a double auction.

MarketURL - The URL of the market service.

Method advertiseRequest returns an identifier of the advertised order. It throws the following exceptions:

InvalidDescriptionException is thrown if the semantic information contained in parameter ClusterRequestDescription is malformed.

InvalidURLException is thrown if either service or market URLs are different than null but are either malformed or not available.

NoSuchAgentException is thrown if an invalid agent identifier is provided.

A.1.2. Application advertisement interface

As mentioned before, (application) services are advertised through their WSDL descriptions. These descriptions are mapped to a domain specific ontology by providing an annotation file (External Annotation File—EAF). The following methods comprise the API for application service advertisement:

Method advertiseService

```
String advertiseService(String WsdlFileLocation, String DomainOntologyNs,
                        String Annotations)
```

This method advertises a service. The specified service is translated to an OWL-S profile, that is registered in the SIS. To produce the OWL-S profile, this method makes use of an external annotations file, whose contents are specified in the *Annotations* String.

`WsdlFileLocation` is the URL of the WSDL document which describes the advertised service

`DomainOntologyNs` is the namespace of the domain ontology which should be used for automatic annotating.

`Annotations` is an XML String with the contents of the annotation file (EAF) for the particular service.

The method returns a unique identifier for the advertisement.

A second form of the advertiseService method is the following:

```
String advertiseService(String WsdlFileLocation, String DomainOntologyNs)
```

In this method, no annotation file (EAF) is submitted by the provider of the service. The SIS automatically performs a matching of WSDL operation message types to ontology classes in the specific ontology.

A.1.3. Market Querying Interface

SIS supports the following types of queries:

- Querying by consumers for markets initiated by providers which are trading tradable resources.
- Querying by providers for markets initiated by consumers who want to purchase tradable resources (reverse markets).
- Querying by providers and consumers for markets initiated by third-party agents.
- Querying for available (application) services by service consumers.

Querying API in the SIS supports operations as described in the following paragraphs.

Method queryMarkets

Method `queryMarkets` performs a query for available markets advertised in the SIS. This method has the following forms:

```
QueryResults[] queryMarkets(ClusterOffer RequestDescription,
                            MarketQuery MarketRelatedConstraints,
                            int NumOfResults,
                            String AgentId) throws NoSuchAgentException,
                                                   InvalidAgentRoleException,
                                                  InvalidDescriptionException
```

This method called by a consumer in order to query for markets advertised by resource provider.

`MarketRelatedConstraints` contains market related constraints in the form intended days, pricing, location, etc.

`RequestDescription` contains resource related specifications for the query.

`NumOfResults` is the number of expected of results. It is used in top-N queries.

`AgentId` is the identifier of the consumer performing the query

Method `queryMarkets` returns an ordered set of endpoint references for markets. It throws

`InvalidAgentRoleException` is thrown when the specified agent identifier does not belong to a consumer agent

`InvalidDescriptionException` is thrown when the Request is malformed

`NoSuchAgentException` - when the submitting agent is not found.

Another form of the queryMarket method is the following

```
QueryResults[] queryMarkets(ClusterOffer RequestDescription,
                            MarketQuery MarketRelatedConstraints,
                            int NumOfResults, String AgentId)
                                       throws NoSuchAgentException,
                                              InvalidAgentRoleException,
                                              InvalidDescriptionException
```

This method called by a provider in order to query for markets advertised by resource consumer.

`MarketRelatedConstraints` contains market related constraints in the form intended days, pricing, location, etc.

`OfferDescription` contains resource related specifications for the query.

`NumOfResults` is the number of expected of results. It is used in top-N queries.

`AgentId` is the identifier of the consumer performing the query

Method `queryMarkets` returns an ordered set of endpoint references for markets. It throws

`InvalidAgentRoleException` is thrown when the specified agent identifier does not belong to a provider agent

`InvalidDescriptionException` is thrown when the Request is malformed

`NoSuchAgentException` is thrown when the submitting agent is not found.

Method queryProviders

A set of providers is returned by this query. The providers can then be invited in a reverse auction market.

```
String[] queryProviders(String RequestId,
                                  int NumOfResults,
                                  String AgentId)
                                  throws NoSuchAgentException,
                                         InvalidAgentRoleException,
                                         NoSuchDescriptionException
```

`RequestId` is the identifier of a resource request, which has been previously advertised in the SIS.

`NumOfResults` is the number of expected of results. It is used in top-N queries.

`AgentId` is the identifier of the consumer performing the query

The method returns an ordered set of provider usernames. Exceptions thrown are:

`InvalidAgentRoleException` - when the specified agent identifier does not belong to a consumer

`NoSuchDescriptionException` - when there is no order with the specified RequestId

`NoSuchAgentException` - when the submitting agent is not found.

Method queryConsumers

```
String[] queryConsumers(String OfferId,
                                  int NumOfResults,
                                  String AgentId)
                                  throws NoSuchAgentException,
                                         NoSuchDescriptionException
```

A set of consumers is returned by this query. The consumers are then invited in an auction market. The market itself is NOT specified by means of. e.g. an endpoint reference.

`OfferId` is the identifier of a resource offer, which has been previously advertised in the SIS.

`NumOfResults` is the number of expected of results. It is used in top-N queries.

`AgentId` - is the identifier of the provider performing the query

The method returns an ordered set of consumer usernames. The exceptions thrown are:

`NoSuchDescriptionException` is thrown when there is no order with the specified OfferId.

`NoSuchAgentException` is thrown when the submitting agent is not found.

A.1.4. Application querying interface

Method queryServices

Service querying is performed by method `queryServices`. This method performs a query for application services, based on the specified I/O types.

```
java.lang.QueryResults [] queryServices(java.lang.String domainOntologyNs,
                                         java.lang.String[] inputTypes,
                                         java.lang.String[] outputTypes)
```

`domainOntologyNs` is the ontology namespace for the domain the user is interested about

`inputTypes` is a list of the required input types. These types are the URIs of classes belonging to the domain OWL ontology.

`outputTypes` is a list of the required output types. These types are the URIs of classes belonging to the domain OWL ontology.

Method `queryServices` returns a list of matched services which are stored in the SIS registry in OWL-S form. The service matching algorithm is described in the Service Matchmaking Section.

A.1.5. Agent Management Interface

Method name	Description
registerConsumer	Register by giving username, password and location
registerProvider	idem
updateAgent	Update agent information
deleteAgent	Unsubscribe the agent

A.2. SIS web interface

The SIS web interface enables users to perform the following functions:

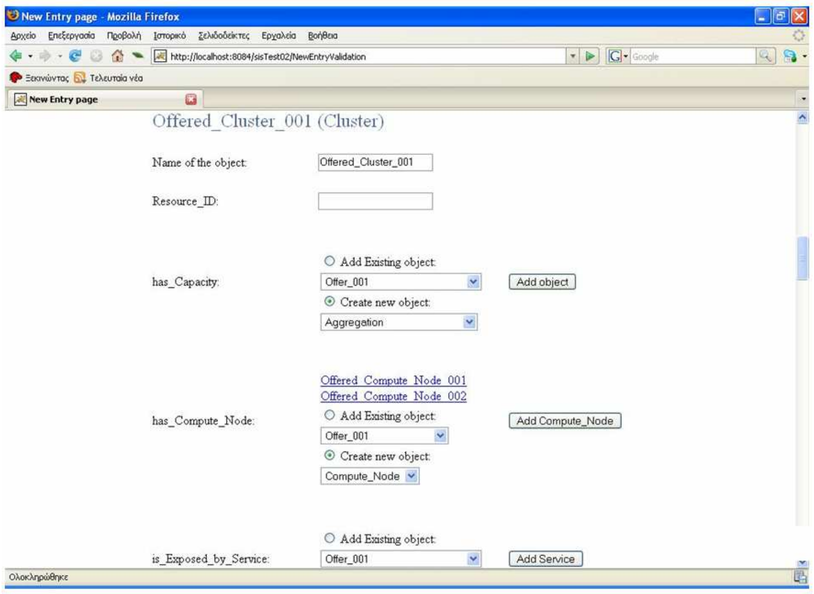
Register an Offer: Providers can use this function to submit new offers, and also to describe the markets where the offers are specified. Using dynamic form creation techniques, users can define offers for already existent registered tradable resources, or register them while creating the offers.

Register a Request: Consumers may use this specific function to create requests for tradable resources, using dynamic forms. Consumers may also impose market or market and order related specifications to further constrain their queries.

Register an object: Apart from the creation of offers or requests, users are capable of specifying ontology classes through this function. The first step here is to define what type of object needs to be registered, by selecting among the various object types (classes) defined in the resource ontology. According to the selected type, a form is generated dynamically so that the user can describe the new entry in detail, avoiding inconsistencies.

Remove an object: Users can use this option to remove registered individuals from the resource ontology.

View registered entries: Here, users can view all their previous individuals and their properties.

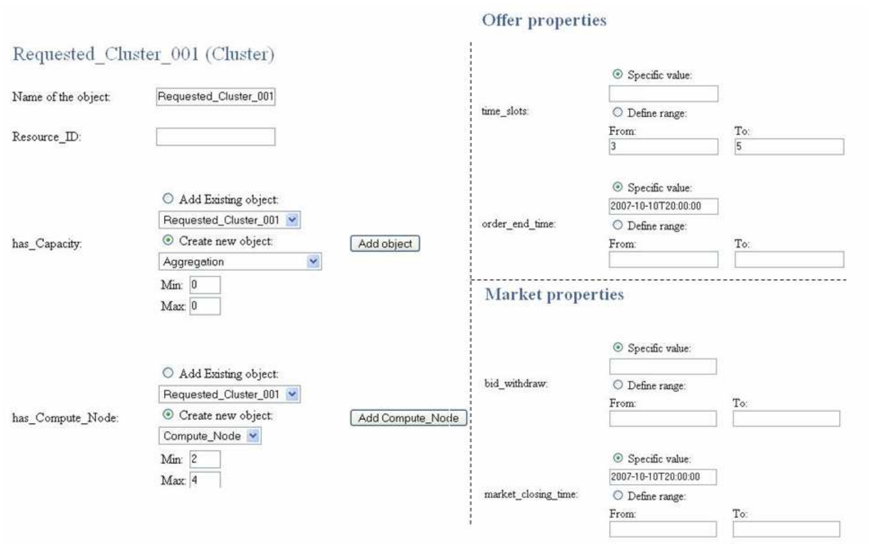

New Entry page - Mozilla Firefox
Αρχείο Επεξεργασία Προβολή Ιστορικό Σελιδοδείκτες Εργαλεία Βοήθεια
Google
Ξεκινώντας Τελευταία νέα
New Entry page
Add object
Add Compute_Node
Add Service
Ολοκληρώθηκε

http://localhost:8084/sisTest02/NewEntryValidation

Offered_Cluster_001 (Cluster)

Name of the object:	Offered_Cluster_001
Resource_ID:	
has_Capacity:	Add Existing object: Offer_001 [✓ Create new object:] Aggregation
has_Compute_Node:	Offered_Compute_Node_001 Offered_Compute_Node_002 Add Existing object: Offer_001 [✓ Create new object:] Compute_Node
is_Exposed_by_Service:	Add Existing object: Offer_001

Figure 10 Offer registration



Requested_Cluster_001 (Cluster)

Name of the object: Requested_Cluster_001

Resource_ID:

has_Capacity: Add Existing object: [✓ Create new object:] Requested_Cluster_001 | Aggregation | Min: 0 | Max: 0 | Add object

has_Compute_Node: Add Existing object: [✓ Create new object:] Requested_Cluster_001 | Compute_Node | Min: 2 | Max: 4 | Add Compute_Node

Offer properties

time_slots: [✓ Specific value:] Define range: From: 3 To: 5

order_end_time: [✓ Specific value:] 2007-10-10T20:00:00 Define range: From: To:

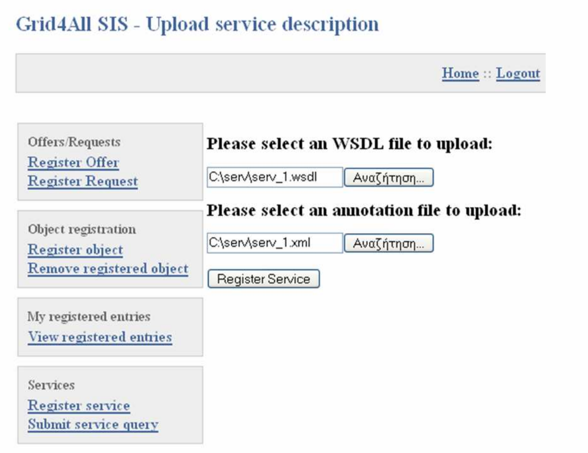
Market properties

bid_withdraw: [✓ Specific value:] Define range: From: To:

market_closing_time: [✓ Specific value:] 2007-10-10T20:00:00 Define range: From: To:

Figure 11 Request registration – Constraints on market-related information

Register service: The registration of a service involves the submission of its WSDL description, as well as the corresponding EAF. Doing so, the OWL-S profile of the WSDL specification is automatically generated, and is stored in the SIS. Using this option, users can register services by providing the required documents.



Grid4All SIS - Upload service description

Home :: Logout

Offers/Requests
Register Offer
Register Request

Object registration
Register object
Remove registered object

My registered entries
View registered entries

Services
Register service
Submit service query

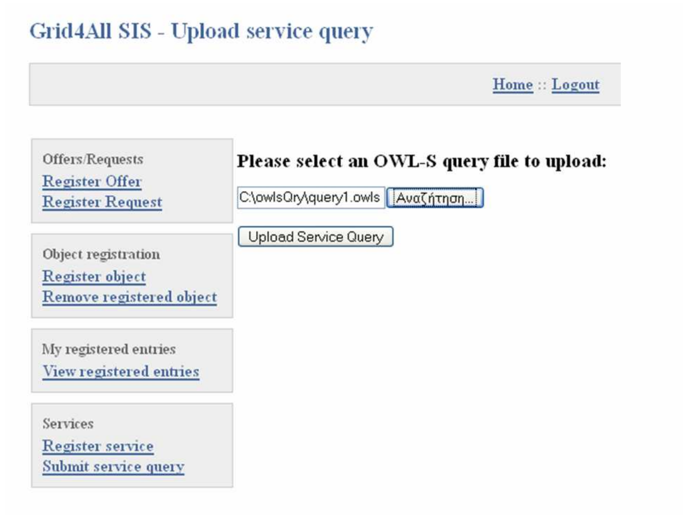
Please select an WSDL file to upload:
C:\serv\serv_1.wsdl Αναζήτηση...

Please select an annotation file to upload:
C:\serv\serv_1.xml Αναζήτηση...

Register Service

Figure 12 Service registration

Submit service query: Service queries, like service advertisements, are encoded using OWL-S. Using this option, users can submit OWL-S profile documents specifying service queries.



Grid4All SIS - Upload service query

Home :: Logout

Offers/Requests
Register Offer
Register Request

Object registration
Register object
Remove registered object

My registered entries
View registered entries

Services
Register service
Submit service query

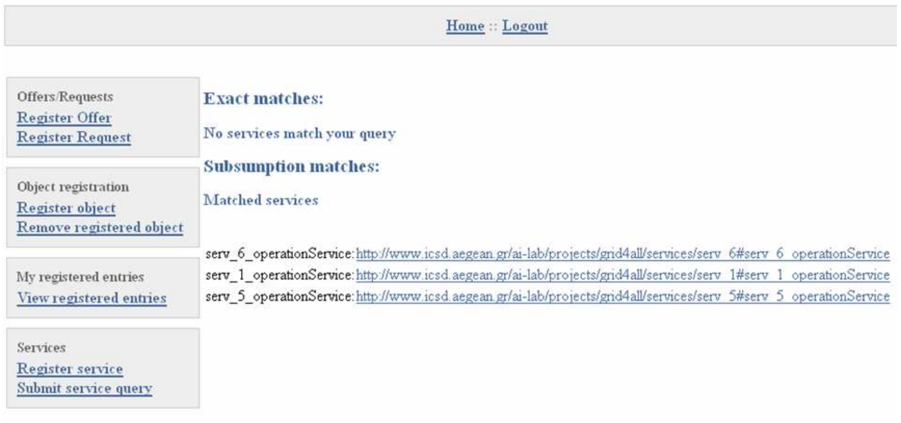
Please select an OWL-S query file to upload:

C:\owlsQry\query1.owls [Αναζήτηση...]

[Upload Service Query]

Figure 13 Service query submission

After such a query is submitted, the matchmaking process takes place and the matched services are presented to the user.



Home :: Logout

Offers/Requests
Register Offer
Register Request

Object registration
Register object
Remove registered object

My registered entries
View registered entries

Services
Register service
Submit service query

Exact matches:

No services match your query

Subsumption matches:

Matched services

serv_6_operationService: http://www.icsd.aegean.gr/ai-lab/projects/grid4all/services/serv_6#serv_6_operationService
serv_1_operationService: http://www.icsd.aegean.gr/ai-lab/projects/grid4all/services/serv_1#serv_1_operationService
serv_5_operationService: http://www.icsd.aegean.gr/ai-lab/projects/grid4all/services/serv_5#serv_5_operationService

Figure 14 Service matchmaking results

A.3. WSDL-Annotation Tool

A.1.6. Dependencies

In order to run WSDL-AT, users need to perform the following actions:

a. WordNet lexicon should be installed. The installation of WordNet should be executed in the same directory with the one specified in the WorldNet configuration file that exists in the WSDL-AT root directory
b. Java version 1.5 or later should be also installed.

A.1.7. Configuration

The WSDL-AT automatically configures itself on startup, by looking for external property files in the application's root directory. The configurable parameters are the following:

a. WSDL elements that will be annotated (annotationElements.properties file)

b. The number of returned suggested mappings of the WSDL-to-OWL mapping process (top-k results) (wsdlAT.properties file)

A.1.8. Use case examples

The WSDL-AT starts by executing “wsdlAnnotationToolRun.bat” batch execution file under Windows OS or by typing the command “java –Xmx81M -jar WSDLAnnotationTool.jar". The GUI of the WSDL-AT consists of two tabs. The first tab named “EAF Editor” is depicted in ***Figure 15***. The ontology can be selected by a pre-defined list using a compo box. Then the WSDL file that will be annotated is selected through the “Browse…” button which opens a browsing window at users’ file system.

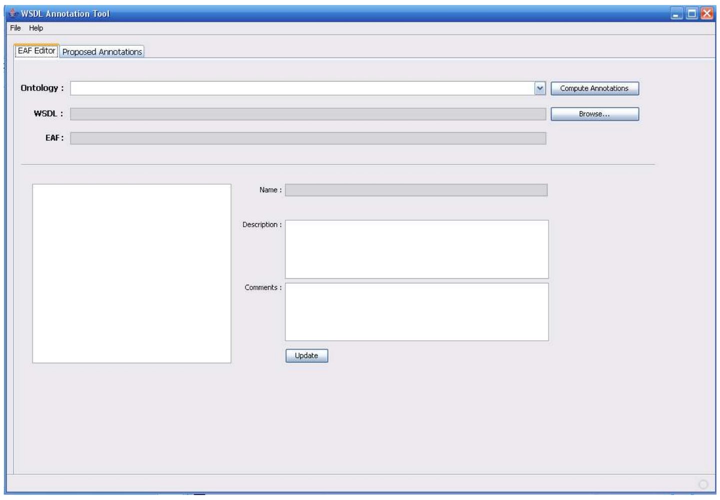
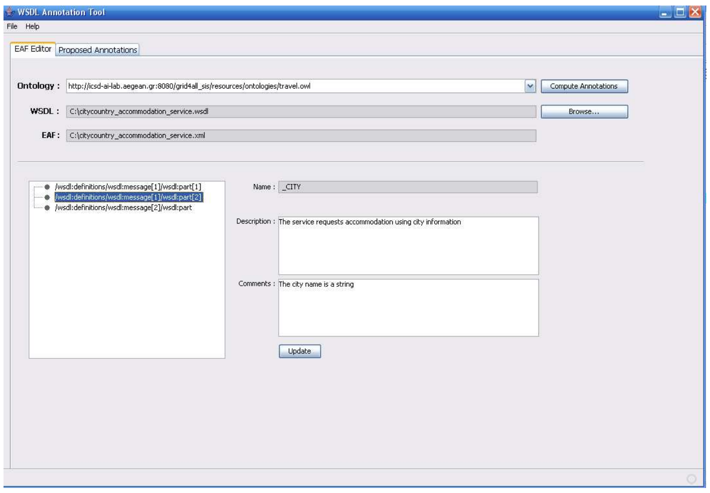


Figure 15 The “EAF Editor” tab

After selecting the WSDL file, its corresponding EAF file is loaded if already exists, otherwise a new one is created by consulting the configuration file specifying the WSDL elements to be annotated. Lists of the WSDL elements that are ready for annotation in their XPATH format are presented to the user. The WSDL element for which the human annotator will provide annotations is selected from this list. Next to this list, there are three text boxes. In the first one from the top, as shown in ***Figure 16***, the value of the “name” attribute of the “part” WSDL element appears (is the one that the XPATH corresponds). The following two text boxes hold annotation information concerning the fields “Description” and “Comments” respectively. Using the “Update” button the human annotator updates the values of these fields. The “Update” button should be pressed before the annotator selects a new WSDL element to be annotated. ***Figure 16*** depicts the annotation of the "/wsdl:definitions/wsdl:message[1]/wsdl:part[2]" WSDL element which corresponds to the “part” WSDL element that its “name” attribute has value equals to “_CITY”.

The human annotator loads the domain ontology by selecting it from the drop down list that is located beside the “Ontology” label. The selection of the ontology is an obligatory action before WSDL-to-OWL mapping process takes place for producing the suggested semantic annotations. The WSDL-to-OWL mapping process is initiated by pressing the “Compute Annotations” button.



WSDL Annotation Tool

Ontology : http://icsd-ai-lab.aegean.gr:8080/grid4all_sis/resources/ontologies/travel.owl

WSDL : C:\citycountry_accommodation_service.wsdl

EAF : C:\citycountry_accommodation_service.xml

- /wsdl:definitions/wsdl:message[1]/wsdl:part[1]
- /wsdl:definitions/wsdl:message[1]/wsdl:part[2]
- /wsdl:definitions/wsdl:message[2]/wsdl:part

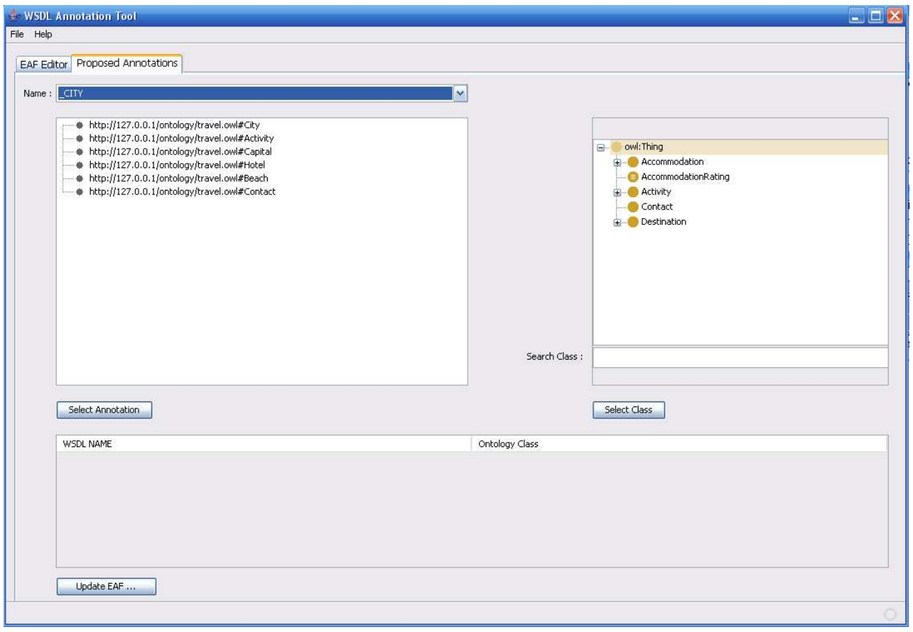
Name : _CITY

Description : The service requests accommodation using city information

Comments : The city name is a string

Figure 16 Annotating the "/wsdl:definitions/wsdl:message[1]/wsdl:part[2]" WSDL element

At the end of the WSDL-to-OWL mapping process the active tab changes to the "Proposed Annotations" tab as ***Figure 17*** depicts. For each WSDL part element the suggested ontology classes are proposed in the left-hand list. Any WSDL part element can be selected from the drop down list of the combo box. The human annotator can select from the list with the suggested ontology classes the correct one, according to which class he/she believes is the correct one. "Select Annotation" button confirms the user-selected semantic annotation for a WSDL part element. The annotator may also select an ontology class directly from the ontology hierarchy that appears in this tab. By pressing the "Select Class" button he/she can replace the ontology class reference of a selected WSDL/Ontology Class pair with a new class. All the selections are kept in a table in the same tab as can been seen in ***Figure 18***. Finally, the EAF is updated with the semantic annotations by pressing the "Update EAF" button.



WSDL Annotation Tool

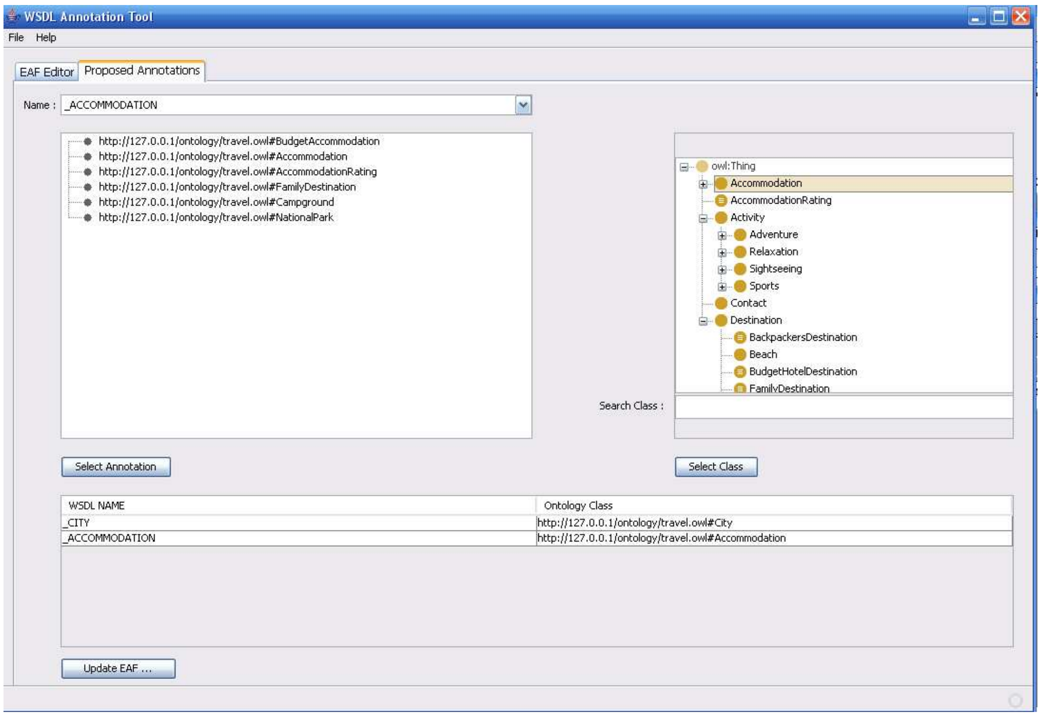
Name : _CITY

- http://127.0.0.1/ontology/travel.owl#City
- http://127.0.0.1/ontology/travel.owl#Activity
- http://127.0.0.1/ontology/travel.owl#Capital
- http://127.0.0.1/ontology/travel.owl#Hotel
- http://127.0.0.1/ontology/travel.owl#Beach
- http://127.0.0.1/ontology/travel.owl#Contact

- owl:Thing
  - Accommodation
  - AccommodationRating
  - Activity
  - Contact
  - Destination

WSDL NAME	Ontology Class

Figure 17 "Proposed Annotations" tab and domain ontology hierarchy



Name : _ACCOMMODATION

- http://127.0.0.1/ontology/travel.owl#BudgetAccommodation
- http://127.0.0.1/ontology/travel.owl#Accommodation
- http://127.0.0.1/ontology/travel.owl#AccommodationRating
- http://127.0.0.1/ontology/travel.owl#FamilyDestination
- http://127.0.0.1/ontology/travel.owl#Campground
- http://127.0.0.1/ontology/travel.owl#NationalPark

Search Class :

WSDL NAME	Ontology Class
_CITY	http://127.0.0.1/ontology/travel.owl#City
_ACCOMMODATION	http://127.0.0.1/ontology/travel.owl#Accommodation

Figure 18 Selected pairs of WSDL elements and Ontology Classes from the suggested ones

After selecting the WSDL file, its corresponding EAF file is loaded if already exists, otherwise a new one is created by consulting the configuration file specifying the WSDL elements to be annotated.

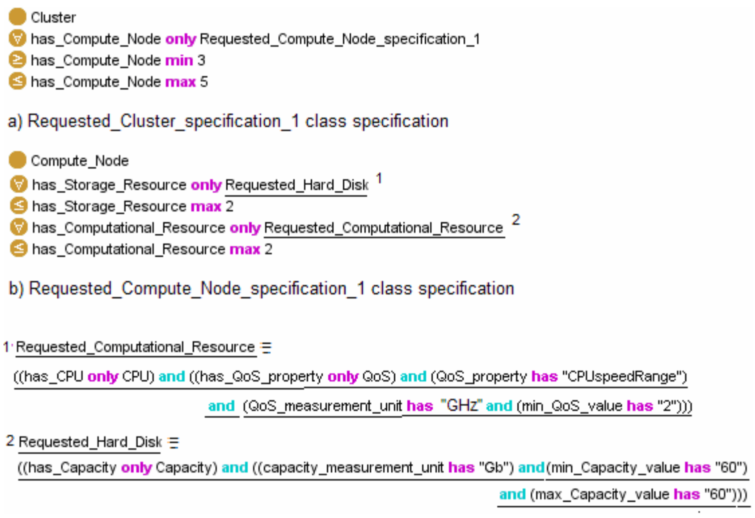
A.4. Markets matchmaking

To demonstrate the retrieval of the traded resources requested by a consumer, we provide details of the classes/individuals specifications (using Protégé interface for presentation reasons) concerning the following request scenario:

A consumer agent places the following request:

> **Resource specification:** *Clusters that comprise at least 3 and at most 5 Compute Nodes, each of which comprises at most 2 CPUs of CPU-speed at least 2 GHz, and at most 2 persistent storage (Hard-Disk) of 60GB.*
>
> **Order (request) constraints:** *At least 1 and at most 3 instances of the requested Cluster type are required for 4 time-slots, each of 30 minutes duration. The Cluster must be offered between 10:00 and 18:00 of the 26th of March, 2007. The maximum price of the Cluster should not exceed 2€ per time-slot.*
>
> The specific **market-related constraints** related to buyer's requests are as follows: *Provider-initiated markets must be located in Athens trading one of the described resources, operating a combinatorial auction, using IC (Incentive-Compatible) as the pricing scheme, and allowing withdraw of orders.*

Figure 19 depicts the specifications of a requested Cluster and Compute Node, as these were described in the matchmaking scenario: "Clusters that comprise at least 3 and at most 5 Compute Nodes, each of which comprises at most 2 CPUs of CPU-speed at least 2 GHz, and at most 2 persistent storage (Hard-Disk) of 60GB".



```
Cluster
has_Compute_Node only Requested_Compute_Node_specification_1
has_Compute_Node min 3
has_Compute_Node max 5
```

a) Requested_Cluster_specification_1 class specification

```
Compute_Node
has_Storage_Resource only Requested_Hard_Disk 1
has_Storage_Resource max 2
has_Computational_Resource only Requested_Computational_Resource 2
has_Computational_Resource max 2
```

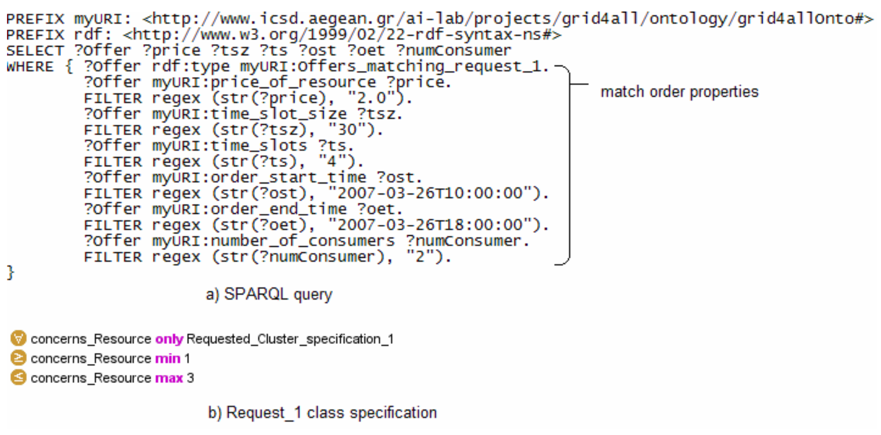
b) Requested_Compute_Node_specification_1 class specification

```
1 Requested_Computational_Resource ≡
  ((has_CPU only CPU) and ((has_QoS_property only QoS) and (QoS_property has "CPUspeedRange")
                          and (QoS_measurement_unit has "GHz" and (min_QoS_value has "2")))

2 Requested_Hard_Disk ≡
  ((has_Capacity only Capacity) and ((capacity_measurement_unit has "Gb") and(min_Capacity_value has "60")
                                                             and (max_Capacity_value has "60")))
```

Figure 19 The requested (a) Cluster and (b) Compute Node specifications. Hard Disk and CPU QoS properties are also depicted (underlined descriptions).

Given the above specifications and the specific Cluster instances retrieved, ***Figure 20*** depicts a specific request: "At least 1 and at most 3 instances of the requested Cluster type are required for 4 time-slots, each of 30 minutes duration. The Cluster must be offered between 10:00 and 18:00 of the 26th of March, 2007. The maximum price of the Cluster should not exceed 2€ per time-slot". The request is done by (a) specifying a subclass of the class Request for the classification of the matching offers, and by (b) a SPARQL query for filtering all the matched offers according to order-related constraints[19].



```
PREFIX myURI: <http://www.icsd.aegean.gr/ai-lab/projects/grid4all/ontology/grid4allOnto#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?Offer ?price ?tsz ?ts ?ost ?oet ?numConsumer
WHERE { ?Offer rdf:type myURI:Offers_matching_request_1.
        ?Offer myURI:price_of_resource ?price.
        FILTER regex (str(?price), "2.0").
        ?Offer myURI:time_slot_size ?tsz.
        FILTER regex (str(?tsz), "30").
        ?Offer myURI:time_slots ?ts.
        FILTER regex (str(?ts), "4").
        ?Offer myURI:order_start_time ?ost.
        FILTER regex (str(?ost), "2007-03-26T10:00:00").
        ?Offer myURI:order_end_time ?oet.
        FILTER regex (str(?oet), "2007-03-26T18:00:00").
        ?Offer myURI:number_of_consumers ?numConsumer.
        FILTER regex (str(?numConsumer), "2").
}
```

a) SPARQL query

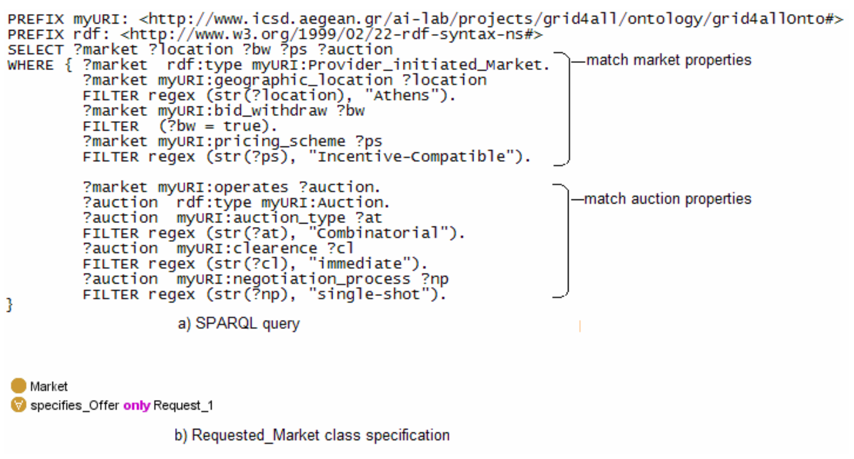
```
concerns_Resource only Requested_Cluster_specification_1
concerns_Resource min 1
concerns_Resource max 3
```

b) Request_1 class specification

Figure 20 The specification of the class Request_1 (b) for the retrieval of offers that specify resources matching with the Requested_Cluster_specification_1 cluster specification, and the SPARQL query (a) for filtering the retrieved offers using the specified request constraints.

Figure 21 depicts the query for the retrieval of the market instances that trade the specific Clusters retrieved through the queries specified above and that have been related with the retrieved offers (retrieved by answering the queries in Figure 8). This is done by exploiting the specific market-related constrains that the buyer agent specified in order to retrieve markets: *"Find any provider-initiated markets located in Athens that trade one of the described resources, such that they operate a Combinatorial auction, the pricing scheme of the market is IC (Incentive-Compatible), and the market allows withdraw of orders."*

[19] In order to reduce complexity of the presentation we do not specify in detail the filtering process, which involves the automatic creation of further ontology classes.



```
PREFIX myURI: <http://www.icsd.aegean.gr/ai-lab/projects/grid4all/ontology/grid4allOnto#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?market ?location ?bw ?ps ?auction
WHERE { ?market  rdf:type myURI:Provider_initiated_Market.
        ?market myURI:geographic_location ?location
        FILTER regex (str(?location), "Athens").
        ?market myURI:bid_withdraw ?bw
        FILTER  (?bw = true).
        ?market myURI:pricing_scheme ?ps
        FILTER regex (str(?ps), "Incentive-Compatible").

        ?market myURI:operates ?auction.
        ?auction  rdf:type myURI:Auction.
        ?auction  myURI:auction_type ?at
        FILTER regex (str(?at), "Combinatorial").
        ?auction  myURI:clearence ?cl
        FILTER regex (str(?cl), "immediate").
        ?auction  myURI:negotiation_process ?np
        FILTER regex (str(?np), "single-shot").
}
```

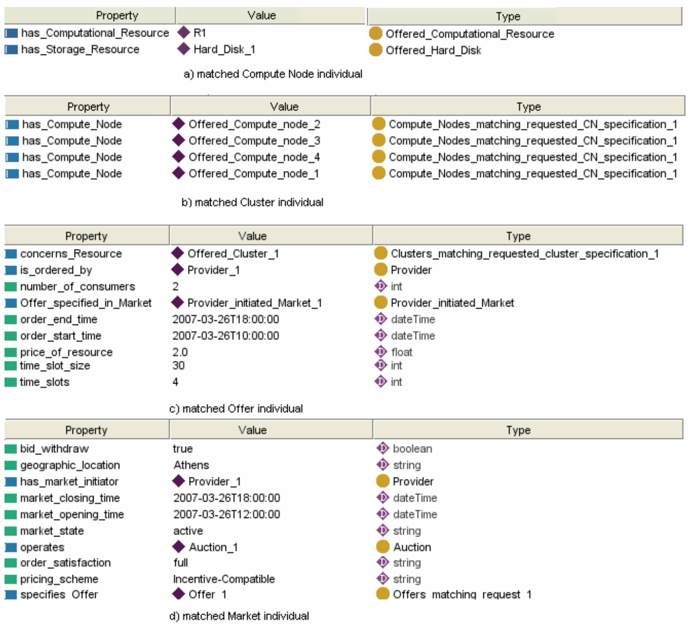
a) SPARQL query

● Market
specifies_Offer **only** Request_1

b) Requested_Market class specification

Figure 21 The Requested_Market class (b) for retrieving the markets that are related with the offers classified under the class Request_1 and the SPARQL query (a) for filtering the retrieved provider-initiated Markets, using certain market and auction constraints.

Running the retrieval process, SIS retrieves the following individuals as presented in Figure 22 a) A Compute Node individual Offered_Compute_node_1 that is part of the requested Cluster, b) a matching Cluster Offered_Cluster_1 c) a matching Offer Offer_1 for the specified Cluster, and d) a matching provider-initiated Market Provider_initiated_Market_1, trading the individual Offered_Cluster_1 offered by the specific Offer_1.



Property	Value	Type
has_Computational_Resource	R1	Offered_Computational_Resource
has_Storage_Resource	Hard_Disk_1	Offered_Hard_Disk

a) matched Compute Node individual

Property	Value	Type
has_Compute_Node	Offered_Compute_node_2	Compute_Nodes_matching_requested_CN_specification_1
has_Compute_Node	Offered_Compute_node_3	Compute_Nodes_matching_requested_CN_specification_1
has_Compute_Node	Offered_Compute_node_4	Compute_Nodes_matching_requested_CN_specification_1
has_Compute_Node	Offered_Compute_node_1	Compute_Nodes_matching_requested_CN_specification_1

b) matched Cluster individual

Property	Value	Type
concerns_Resource	Offered_Cluster_1	Clusters_matching_requested_cluster_specification_1
is_ordered_by	Provider_1	Provider
number_of_consumers	2	int
Offer_specified_in_Market	Provider_initiated_Market_1	Provider_initiated_Market
order_end_time	2007-03-26T18:00:00	dateTime
order_start_time	2007-03-26T10:00:00	dateTime
price_of_resource	2.0	float
time_slot_size	30	int
time_slots	4	int

c) matched Offer individual

Property	Value	Type
bid_withdraw	true	boolean
geographic_location	Athens	string
has_market_initiator	Provider_1	Provider
market_closing_time	2007-03-26T18:00:00	dateTime
market_opening_time	2007-03-26T12:00:00	dateTime
market_state	active	string
operates	Auction_1	Auction
order_satisfaction	full	string
pricing_scheme	Incentive-Compatible	string
specifies_Offer	Offer_1	Offers_matching_request_1

d) matched Market individual

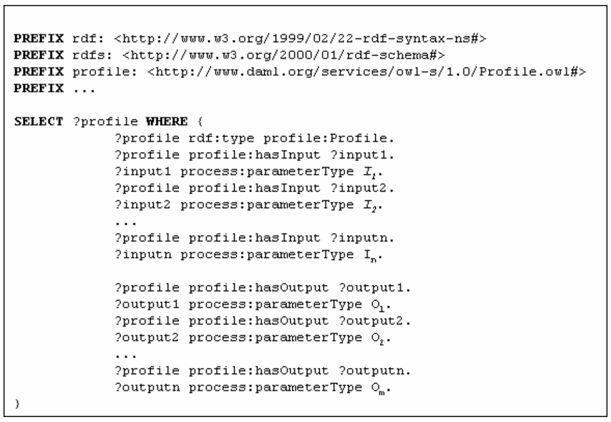
Figure 22 Starting from the bottom of the Figure: (d) The matched provider-initiated market individual, and (c) the related matched offer which trades the matched Cluster specification shown in (b), which, in its own turn, comprises the specification of the matched Compute Node shown in (a).

A.1.9. Services Matchmaking

An example of the various possible matching types is presented in Table 1. The I/O types used in the example are taken from the Grid4All resource ontology. Tradeable_Resource subsumes Compute_Node, Hardware_Resource subsumes CPU and Hard_Disk

Service query specifications:			
Input types: Compute_Node			
Output types: Hardware_Resource			
Advertised Service #	**Input types**	**Output types**	**Type of match**
1	Tradeable_Resource	CPU	Subsumes
2	Tradeable_Resource	Provider_initiated_Market	Fail
3	Market	Tradeable_Resource	Fail
4	Tradeable_Resource	Consumer_initiated_Market	Fail
5	Tradeable_Resource	Hard_Disk	Subsumes
6	1. Tradeable_Resource 2. Market	CPU	Subsumes
7	Compute_Node	Hardware_Resource	Exact

Table 1. Service matching example



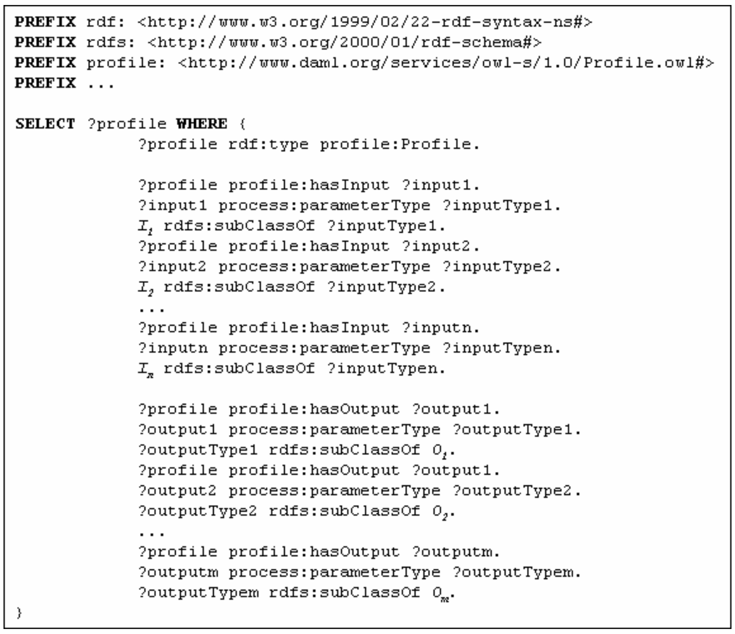
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX profile: <http://www.daml.org/services/owl-s/1.0/Profile.owl#>
PREFIX ...

SELECT ?profile WHERE {
            ?profile rdf:type profile:Profile.
            ?profile profile:hasInput ?input1.
            ?input1 process:parameterType I1.
            ?profile profile:hasInput ?input2.
            ?input2 process:parameterType I2.
            ...
            ?profile profile:hasInput ?inputn.
            ?inputn process:parameterType In.

            ?profile profile:hasOutput ?output1.
            ?output1 process:parameterType O1.
            ?profile profile:hasOutput ?output2.
            ?output2 process:parameterType O2.
            ...
            ?profile profile:hasOutput ?outputn.
            ?outputn process:parameterType Om.
}
```

Figure 23 SPARQL query for service matching (Exact match)

Figure 23 and Figure 24 present the SPARQL queries which have been created according to a given query to perform the matchmaking at the service profile level. Granted that the query specifies n input parameters of types I_1, I_2, …, I_n, and m outputs 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 type defined in the service query.



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX profile: <http://www.daml.org/services/owl-s/1.0/Profile.owl#>
PREFIX ...

SELECT ?profile WHERE {
            ?profile rdf:type profile:Profile.

            ?profile profile:hasInput ?input1.
            ?input1 process:parameterType ?inputType1.
            I1 rdfs:subClassOf ?inputType1.
            ?profile profile:hasInput ?input2.
            ?input2 process:parameterType ?inputType2.
            I2 rdfs:subClassOf ?inputType2.
            ...
            ?profile profile:hasInput ?inputn.
            ?inputn process:parameterType ?inputTypen.
            In rdfs:subClassOf ?inputTypen.

            ?profile profile:hasOutput ?output1.
            ?output1 process:parameterType ?outputType1.
            ?outputType1 rdfs:subClassOf O1.
            ?profile profile:hasOutput ?output1.
            ?output2 process:parameterType ?outputType2.
            ?outputType2 rdfs:subClassOf O2.
            ...
            ?profile profile:hasOutput ?outputm.
            ?outputm process:parameterType ?outputTypem.
            ?outputTypem rdfs:subClassOf Om.
}
```

Figure 24 SPARQL query for service matching ("Subsumes" match)

For the second type of service matching, the “subsumes” type of match, the corresponding SPARQL query Figure 24, is created so that a matching service will be recognized if a) each one of its input parameters subsumes an input type which is defined in the service query b) each one of its output parameters is subsumed by an output type which is defined in the service query. To process such a query, which uses the subclassOf property of the RDF Schema, it is necessary to use an inference engine, so that inferred subclass/superclass relations may be detected among the parameter types specified in service queries or advertisements.

B. Resource Broker

B.1. Auction server implementation

The CAS architecture proposes a set of interfaces to control and manage sub-component states and a set of primitives to handle market specific *events*, *timers,* and *state* management. Each composite component, i.e., a component that includes more than one primitive component is expected to implement the *ActivityController* interface (as a managing component) and each contained sub-component is expected to implement the *ActivityControl* interface (as a managed component). The Figure 25 depicts the state transitions of the sub-component *Auction.* State transitions may be triggered due to events such as timers, method invocations or control actions from parent states. In this figure the Auction component inherits the states of its parent component and manages its own sub-states as well (in bidding, clearing etc). Manager components implementing the *ActivityController* interface propagate synchronization and events to signal state changes to its sub-components. The Figure 26 illustrates a sequence of activity that is part of initialization of a newly deployed auction market. Here the initiator of the market invokes the *MarketControl* interface to start market operations. This triggers the change of state to *OPEN* which will then be propagated to the hierarchy of sub-components through the *ActivityControl* interface. A sub-component changes to the new compound state when it's including component triggers this change and when its current sub-state permits it; e.g. the *Registration* component is open to accept registrations only when the Market is open and when the registration start timer (if configured) has expired.

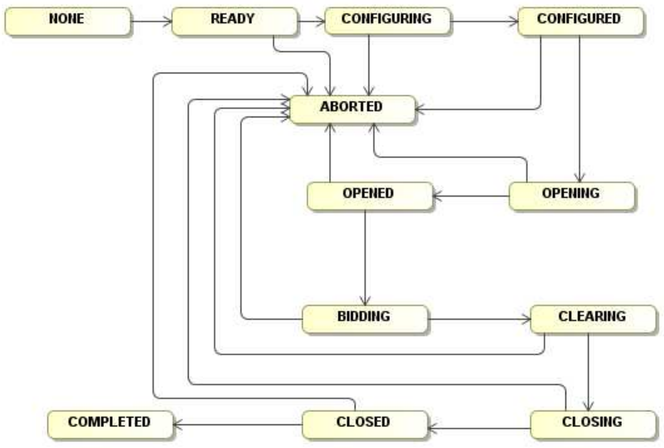


Figure 25 Auction state machine

We have implemented workflows for two patterns: single shot auctions and iterative auctions. Fractal Controller interfaces (Content, Binding) are used to introspect sub-components and interfaces to verify that every composite sub-component indeed respects the programming constraints; that the Controller and Control interfaces are indeed implemented and bound according to the architecture as has been defined in the auction's ADL description.

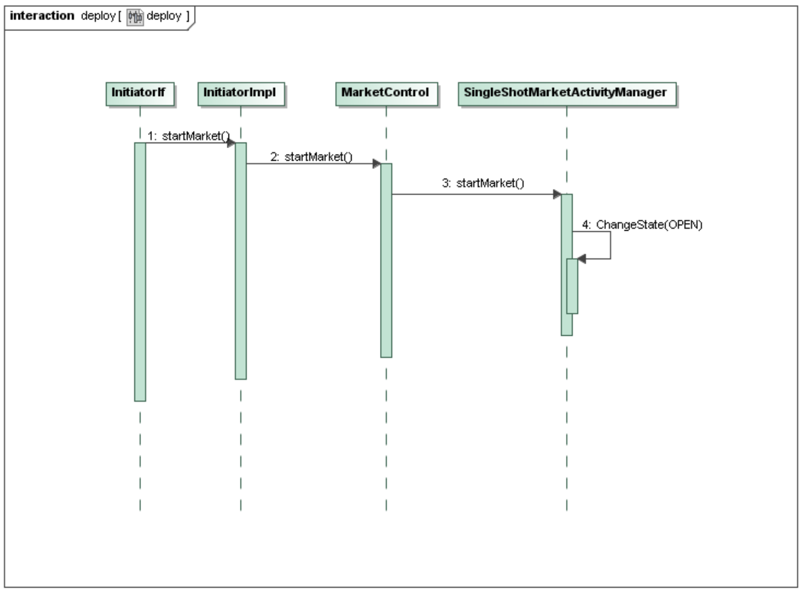


Figure 26 Initiator interface example

We have implemented and evaluated the K-pricing based double auction using the framework described within this section. The next section describes the internals of this mechanism and its relevance to auctioning computational resource leases.

The architecture of the configurable auction server is described declaratively using the Fractal Architecture Description Language. The ADL run-time software is capable of deploying and configuring the application according to the specification provided by this description. The ADL describes the architecture as a hierarchical set of components, the interfaces offered and provided by each component, and the bindings between the interfaces (provided/offered). The Fractal ADL also has run-time support that is provided by the Fractal ADL factory, which can be used to deploy the described architecture. The boxes below show a Java code snippet illustrating deployment and a fragment of the ADL.

```
public void deployerMarket(String auctionADL) throws MarketException {
    try {
       factory =
       FactoryFactory.getFactory(FactoryFactory.FRACTAL_BACKEND);
       marketServer = (Component) factory.newComponent(adlDef, context);
    } catch (Exception e) {
       throw new Error("Cannot get Fractal ADL factory", e);
    }
}
```

```
<!-- Main market server application that is, the definition -->
<definition name="server.lib.MarketServer" extends = "MarketServerType">
    <component name="market-server" definition = "MarketServerManager"/>
        <component name="market" definition="market.lib.Market"/>
           <binding client="this.Market server" server="market-server.Market server"/>
            <binding client="market-server.Market event dispatcher" server="market.Market
event dispatcher"/>
</definition>
```

Figure 27 ADL fragment for Market

The Figure 27 shows an excerpt of an ADL. Specific implementation class of a component may be configured through the ADL. Different market/auction formats (implementing specific auction rules) may be deployed by simply modifying such *Content* classes. Deployment code registers the main client interfaces at RMI registries; clients look up the *MarketServer* interface which acts as an entry point to connect and register to the auction.

D2.2 describes the extensions and enhancements that have been done to the Fractal ADL in order to support deployment of application components within virtual organisations. With these extensions, finer control may be done on the placement of components on a network of compute nodes. At the current prototype of the auction server, we are not using these extensions leaving the use of them to our future work.

B.2. K-Double auction design and evaluation

The K-DA mechanism implements specific interfaces of the architecture described in section 5.4.2. At initialization time, this can be configured to execute either in mode *'continuous'* or *scheduled*. In case of continuous clearing, clearing can be activated by different events: at each new bid, when a configured number of bids have been accepted etc.

The K-pricing based double auction is a simple yet powerful mechanism to trade in multiple units of single items; an item could be CPU or Storage. This section describes how the implementation of this mechanism fits into the previously described architecture and the specific extensions to this well known mechanism to adapt to trading in multiple time-slots. The native mechanism is capable of trading multiple units of an item, but does not distinguish one item to the other. This mechanism however does not guarantee complete allocations and hence does not accept bids that specify the AND operator described previously.

Leasing computational resources imply that the time is divided into discrete intervals called *time-slots*. The intervals need not be necessarily of the same size, but we assume this for convenience. The standard DA mechanism cannot be directly used since there may be multiple CPUs and multiple time-slots for each traded CPU. We hence extend the standard mechanism to support multiple time-slots; e.g., a CPU offered between 12:00 and 18:00 is traded as 6 time-slots of one hour each. We allow consumers to place bids for more than one CPU unit and for more than one time-slot: *"2 cpus for 2 hours between 12:00 and 18:00 for 1$ for each cpu for each time-slot"*. Such bids are referred to as *substitutes*, since they are willing to accept any contiguous subset of time-slots in the specified interval. These are implicit XOR bids and a pre-processor expands them to all valid combinations.

The *BidCatalog* component uses and extends the four heap algorithm as described in [CAS-1]. It maintains the multiple 4Heap data structures; this is referred to as a *4HeapBidLaneEngine* and one such lane is maintained for each time-slot. Bids requesting more than one time-slot are dispatched across the lanes. Now consider the following two bids: B1 that requires 1 CPU for any two time-slots between 12:00 and 18:00 and B2 that precisely requires the time-slot 13:00. *What we would like is that if the offered prices are compatible, then B1 should not pre-empt B2 for the time-slot at 13:00.*

When handling such XOR bids, we considered two possibilities: (a) Select one lane where the bid can currently win and choose only that XOR leaf-node. (b) Dispatch the XOR bids to every acceptable lane. Currently we implement the second option. In the case of XOR bids, only one branch should win (exclusive OR). Lanes are cleared in random orders and the first winner removes its siblings from all the other lanes.

We have revised the above naive algorithm:

- Dispatch an XOR bid to the lane (time-slot) where it maximizes social welfare.
- Implement clearing in multiple passes; at end of each pass, an XOR bid is selected as a winner on the lane where it produces the greatest welfare. The clearing iterates until there are no more XOR siblings to remove.

The Figure 28 gives the main classes that constitute the specific implementation support for this mechanism.

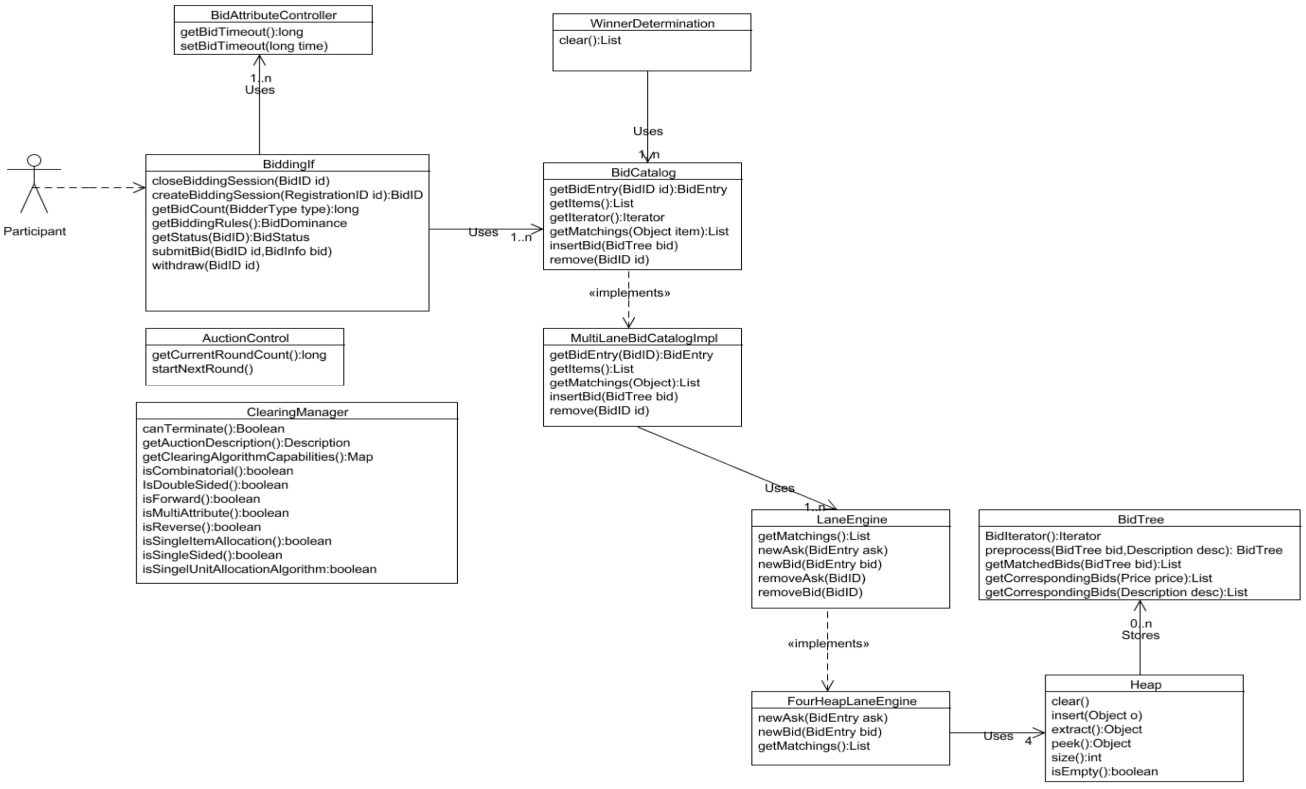


Figure 28 K-DA classes

The implementation uses the Four Heap data structure. The 4-HEAP algorithm [CAS-1] takes its name from the fact that the bids are organized into four heap data structures, representing the currently winning buy offers, the currently winning sell offers, the currently non-winning buy offers, and the currently non-winning sell offers. We denote the four heaps as Bin, Sin, Bout, and Sout, respectively. Bin and Sout are min heaps, and Bout and Sin are max heaps. Naturally, the minimal bid in Bin must be at least as great as the maximal bid in Bout, and the minimal bid in Sout must be at least as great as the maximal bid in Sin. The heaps have the further properties that the number of units in Bin and Sin must be the same, and the minimal member of Sout is strictly less than the maximal member of Bout. An important benefit of the 4-HEAP algorithm is that the Mth and (M + 1)st prices are easily calculated from the heaps. Specifically, the Mth-price is the min of Bin and Sout, and the (M + 1)st-price is the maximal members of Bout and Sin, both of which can be easily computed from the values of the highest priority nodes in each heap. In addition, clearing the auction is simply a matter of deconstructing the Bin and Sin heaps, and leaving the other two heaps intact.

When a new bid is inserted, the algorithm first determines whether it (a) should be inserted directly into an OUT heap, (b) should be matched with bid(s) in the complementary OUT heap and all implicated bids moved to IN heaps, or (c) displace some bids from the appropriate IN heap. In both cases (b) and (c), the new bid may need to be split across the IN and OUT heaps during this process. Split bids can be reassembled if the component parts are returned to the same heap, thus, the algorithm will never have more than one split bid. Note that steps (b) and (c) may result in several nodes being moved between heaps; in the worst case, q nodes will have to be moved, where q is the quantity associated with the new bid.

A price quote can be generated in constant time by simply computing the Mth- and (M + 1)st-prices as mentioned above. Setting aside the issue of how bids are matched, and focusing only on separating the winning bids from the non-winning bids, clearing takes O(N) time in 4- HEAP because the two IN heaps can be directly disassembled. In particular, the 4-HEAP algorithm has the drawback that its typical performance is close to its worst case performance because of the manner in which objects are popped and pushed onto the heap. In 4-HEAP, when a bid is moved from an IN heap to an OUT heap, or vice-versa, it is certain to require two operations (a pop then a push) that require exactly ln(N) time.

Bid pre-processing

The need for pre-processing multi-item bids into single-item bids has been identified. Preprocessing may be looked upon as an internal process that returns semantically equivalent bids but able to be handled by a specific allocation mechanism. There is the need to point out that a given market (instance) regulates the operators that may be present at non-leaf nodes. For example, a market employing an auction that cannot guarantee the complete allocation of the request will not accept the AND operator. The following example will be used to illustrate the different possibilities of decomposition: Let's consider an auction that is trading one CPU of 400 FLOPS for the time range compressed within 9:00 and 19:00 where each time slot is 1 hour of duration. The bids that users are able to formulate are of the following types:

- **Exact preference in quantity and time:** A bid B3 requires one CPU of 400 FLOPS for 3 hours from 12:00 to 15:00, that is, the bidder is asking for a precise time range. Case 1 of Figure 29 shows the compact bid representation that is the way user formulates the bid. Note that for this example bid partial satisfaction is required (OR constraint). Case 2, presents the same case when complete satisfaction is required (AND constraint). The (b) of Figure 30 represents the same bid but showing it fully pre-processed. Figures in this document will present both compact and full decomposition trees except for the cases where the size of the full decomposition prevents from a clear understanding.
- **Exact preference in quantity but not in time:** Require time to be consecutive: A bid B4 requires one CPU of 400 FLOPS for 3 consecutive hours within 11:00 and 16:00. This case requires the formulation of one XOR bid with (number of available slots − number of required slots + 1) AND (OR for the case of partial satisfaction bids) sibling bids each one with 3 precise leaf nodes. Case 1 of Figure 29 shows the bid formulated by the bidder for the case when partial satisfaction is required. Case 2, presents the same case when full satisfaction is required. Figure 29 presents the tree completely pre-processed for the case of partial satisfaction.
- **Exact preference in quantity but not in time:** Do not require time to be consecutive: A bid B5 requires one CPU of 400 FLOPS for 3 any hours within 11:00 and 16:00. This case requires pre-processing the bid into a tree with C_n^n leaf nodes. Due to its size a figure is not provided, however the case is very similar to the one presented in Figure 30.
- **Neither exact preference in quantity nor in time and contiguous time-slots:** By excess: A bid B6 requires one CPU of 400 FLOPS at least for 2 consecutive hours within 11:00 and 16:00. In this case, the auction provides complete satisfaction (the auction only accept AND operators), otherwise does not make sense the mandatory 'at least'. The decomposition needs to generate a bid tree of XOR (root node) and AND due to impreciseness in time (multiple possibilities). The maximum bound on number of time-slots must be set by bidder. Due to the size of the decomposition, Figure 30 only shows the decomposition of non-leaf nodes whereas leaf nodes are kept in a compact representation. For the example, the bids are decomposed into precise bids for 2 time slots, 3 time slots and 4 time slots (bound set by the bidder). The complete decomposition (including leaf nodes) would be similar to the decomposition showed in right side of Figure 31.
- **Neither exact preference in quantity nor in time and contiguous time-slots:** By default: A bid B7 requires one CPU of 400 FLOPS at most for 2 consecutive hours within 11:00 and 16:00. This case is simpler and similar to the case illustrated in Case 1 of Figure 29. Notice that the requirement 'at most' indicates partial satisfaction which is expressed with an OR constraint.

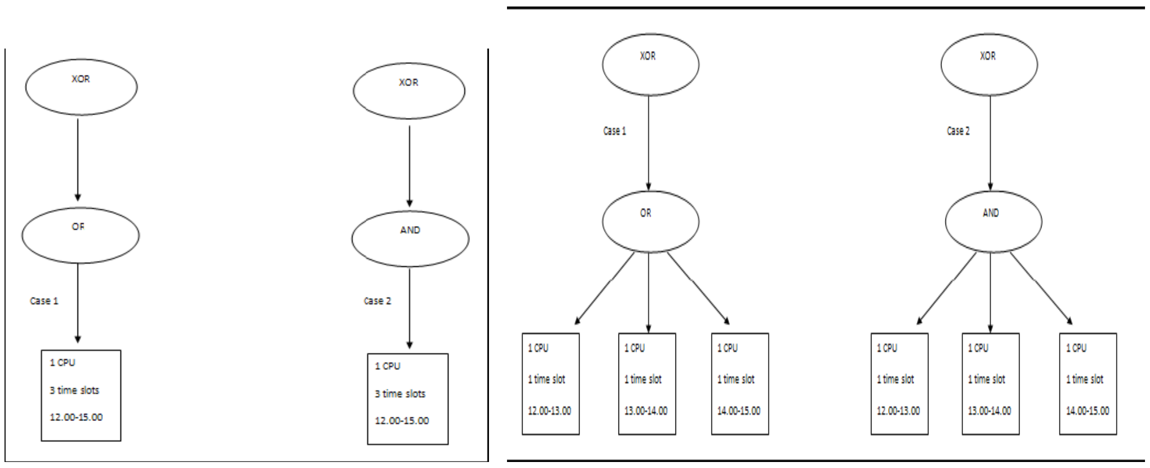


Figure 29 Exact preferences in quantity: (a) Compact representation (b) Complete representation

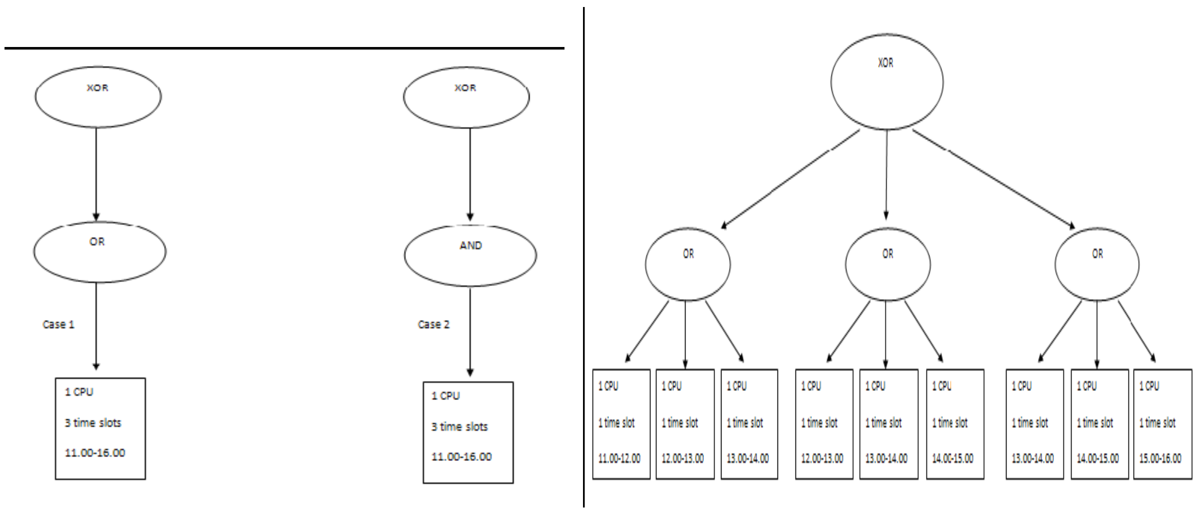


Figure 30Compact and complete representations

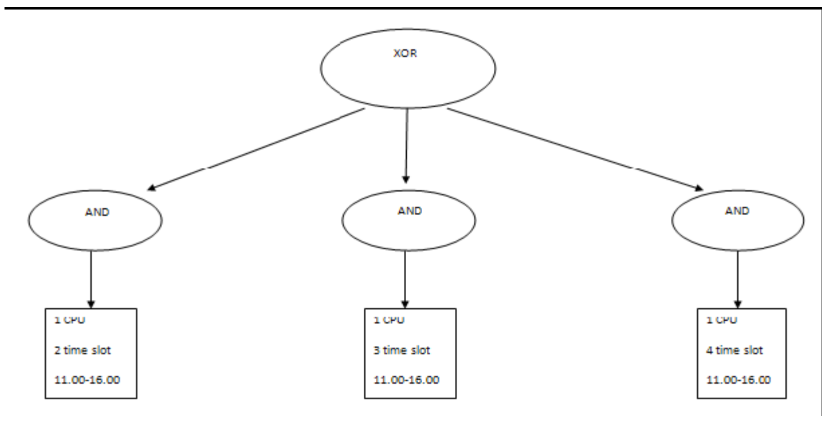


Figure 31 Exact preference in quantity and not time

K-DA Evaluation

This section presents current evaluation results that have been conducted using the K-pricing double auction mechanism implementation. This mechanism is used to allocate time-slots of one type of resource, where a resource may be CPU or storage. Bidders may request (or offer) multiple units, either in the number of CPU units or in the number of time-slots. The objective of the mechanism is to maximize social welfare, i.e., the aggregated sum of utility generated by the allocations. A *Bid* is a translation of the requirements of a *Job* (we use this term following Grid terminologies) that is released by a user: type of resources, quantity of resources, attributes of resources, time specifications and the price. The evaluation studies the effect of different statistical distributions used to generate the different parameters of a Bid on the results of the auction measured by the aggregated welfare and aggregated allocations.

In our model, each bid requires a certain amount of resources for a specified duration. The price of a resource is computed using the following functions:

$$Q_{base} = n_{rec} \cdot t_{rec_i} \qquad \textbf{E 1}$$

$$p_{base} = Q_{base} \cdot p_{avg} \qquad \textbf{E 2}$$

$$p_{res} = p_{base_i} \cdot \tau \qquad \textbf{E 3}$$

$$p_{bid} = \sum_{i=1}^{k} p_{res_i} \qquad \textbf{E 4}$$

$$\tau = (1 - \mathrm{N})^{-1/\varsigma} \qquad \textbf{E 5}$$

Parameter	Description	Distribution	Parameters
n_{rec}	Number of units of required resource	Gaussian	μ, σ
t_{rec}	Number of time slots that this resource is required.	Gaussian	μ, σ
Q_{base}	Quantity of the required resource	N/A	N/A
P_{avg}	Average cost price of the resource across providers.	N/A	N/A
P_{base}	Base price for the quantity of resources required.		
P_{bid}	Sum of the prices of the items required in the bid.		
τ	The willingness to pay of the bidder.	Pareto	ς
p_{sell}	Sell price per unit and per time slot.	Gaussian	μ, σ

Asks are characterized in the same way as bids.

$$p_{avg} = \sum_{i=1}^{k} \frac{p_{sell_i}}{k} \qquad \textbf{E 6}$$

In order to generate test data we assumed that each bid was for a single unit of resource leading to the equality:

$$p_{bid} = p_{res} \qquad \textbf{\textit{E 7}}$$

The generated bids followed a Pareto distribution with two different variable parameters: the Pareto index and the $P_{base.}$ The generated prices had a mean and variance as indicated in equations 8 and 9 below:

$$\mu = \frac{\varsigma \cdot P_{base}}{\varsigma - 1} \qquad \textbf{\textit{E 8}}$$

$$v = \frac{{p_{base}}^2 \cdot \varsigma}{(\varsigma - 1)^2 \cdot (\varsigma - 2)} \quad \textbf{\textit{for (}} \varsigma \textbf{\textit{>2) E 9}}$$

Similarly asks where generated using a Gaussian distribution with mean μ and standard deviation σ .

In order to calculate p_{sell} (the based selling or reservation price per unit of resource), Sun Grid [CAS-3] and Amazon Web Services [CAS-4] pricing models have been studied. Sun Grid applies a fee of 1$ for one hour of CPU. Amazon Web Services uses a different pricing policy. Each request to an Amazon Web Service is charged with 0.000001 $ whereas data transfer are charged with 0, 10$ for each in GB and a variable out rate ranging from (0,18$ GB/month to 0,13$ GB/month). Our p_{sell}, price has been calculated following the model taken by Sun using a mean of 1$ and a standard deviation of 0,125$. Note that Sun used a fixed price policy. However we wanted to model some variability of prices because we supposed that resources were provided by different sellers. To generate random data the SSJ statistical library [CAS-6] has been used. Random numbers have been generated with a backbone generator of the type Well Equidistributed Long period Linear Random Number Generator (WELL), proposed by F. Panneton, and which has a state size of 1024 bits and a period length of approximately 2^{1024}.

Ask have been generated following a *Gaussian* distribution *N* (*μ*, *σ*) with a mean *μ* and variance σ^2, where *σ* > 0. Its density function is

$$f(x) = 1/(2\pi)^{1/2}\sigma e^{(x-\mu)2/(2\sigma 2)}$$

To generate bids, a *Pareto* distribution with parameters ς > 0 and P_{base} > 0 have been used. Its inverse distribution function is

$$\mathrm{F}^{-1}(u) = \mathrm{P}_{\mathrm{base}}(1-u)^{-1/\varsigma} \qquad \text{for } 0 <= u < 1.$$

We have conducted two sets of experiments and describe them in the following sections. At this time we have not correlated the generated data to real and representative work-load traces. The experiments are conducted by injecting the generated work-load to a K-double auction. The arrival rates (of bids and asks) are not relevant to this experiment.

SIGMA Experiments

We aimed to experiment the effects of varying the standard deviation of asks (while keeping unchanged that of bids) in the final results of the auction. Specifically we measured the average price achieved by the auction, the total and average social welfare and the number of transactions that occur. These variables were observed for different values of *σ* in the interval [0,125, *μ*] with a fixed increment of 0.001. Bids were generated with fixed values of ς and P_{base} calculated as (2) with $p_{avg} = \sum_{i=1}^{k} \frac{p_{sell_i}}{k}$ for k sellers.

The studied variables were:

Name	Symbol	Description
Number of successful transactions	N_t	Counts the number of bids that are matched with asks.
Number of issued bids and asks	N_{total}	Counts the number of issued bids in the auction
Avg Final Price	PF_{avg}	The average transaction price for the total amount of transactions for an execution.
Social Welfare	W_t	Accumulated social welfare for all transactions in an execution.
Avg Social Welfare	W_{avg}	$W_{avg}=W_t/N_{total}$
Social Welfare for those bidders (sellers and buyers) that get a match.	W_{alloc}	$W_{avg}=W_t/N_t$
Accumulated Seller's Revenue	R_{ts}	Sum of the revenues of all the sellers.
Accumulated Buyer's Revenue	R_{tb}	Sum of the revenues of all the buyers.
Final Price	P_{final}	It can be Uniform, so all transactions in an experiment occur at the same price. I can be discriminatory, that is, all transactions calculate a price.
K in [0,1]	K	The pricing policy.

Social welfare has been calculated as the accumulation of the revenue obtained by the seller plus the revenue obtained by the buyer.

$$W_t = \sum_{i=1}^{m} P_{bid_i} - P_{sell_i} \quad (10)$$

Average welfare it the total welfare divided to the total amount of issued bids and asks.

$$W_{avg} = \frac{\sum_{i=1}^{m} P_{bid_i} + P_{ask_i}}{N_{total}} \quad (11)$$

W_{alloc} indicates the welfare only considering those bidders that get a match.

$$W_{alloc} = \frac{\sum_{i=1}^{m} P_{bid_i} + P_{ask_i}}{N_t} \quad (12)$$

R_{ts} is the total revenue for sellers while R_{tb} represents the total revenue for buyers. Note that social welfare is computed as the sum of R_{ts} and R_{tb}.

$$R_{ts} = \sum_{i=1}^{m} P_{final} - P_{sell} \quad (13)$$

$$R_{tb} = \sum_{i=1}^{m} P_{bid} - P_{final} \quad (14)$$

The experiments consisted in 875 executions of the clearing process each one with a different ask input. For each experiment the same 100 bids were used. In contrast, 100 asks were generated as described above for each experiment. Characteristic parameters for bid generation were kept constant while characterization parameters for asks generation were changed at each experiment. For bids, we used a fixed value of ς set to 1.2 and a P_{avg} calculated as the average value of asks. Asks were generate starting with an σ value of 0.125 and increasing it by 0.001 after each experiment. μ was fixed at 1.0 as described earlier. The K-pricing policy used in these experiments set the value of K to 0.5.

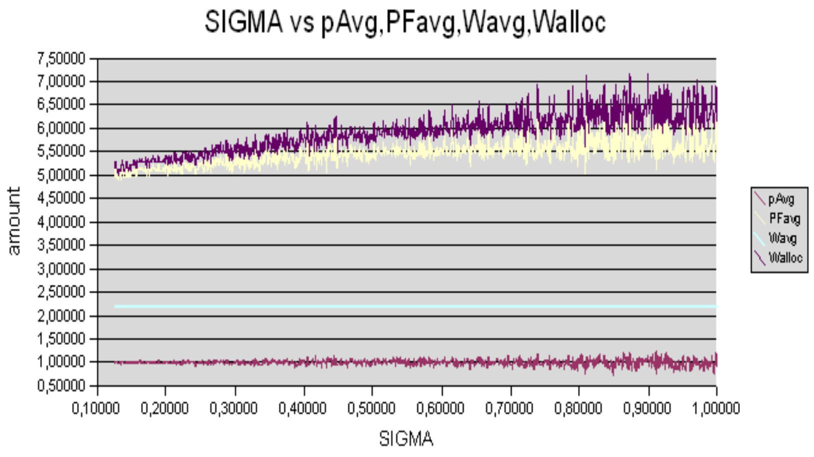


Figure 32

Figure 32 presents the effects of varying σ (standard deviation) to PF_{avg}, W_{avg}, P_{avg} and W_{alloc}. We can see that as σ increases, the variability of P_{avg} also increases (at it is expected) but the average base price falls quite near to the 1$. Note that this variability can be also attributed to the imprecision of the used random generator [5].

The final average price (PF_{avg}) slightly increases as σ increases due to the fact that as more variability, sell prices can be greater leading to a higher final average price. Accordingly, as PF_{avg} increases, the W_{alloc} value also increases because there is more benefit to share (more variability lead to smaller sell prices). W_{avg} is kept almost constant (increases very insignificantly) since the price increment is compensated by the decrement of the number of allocations.

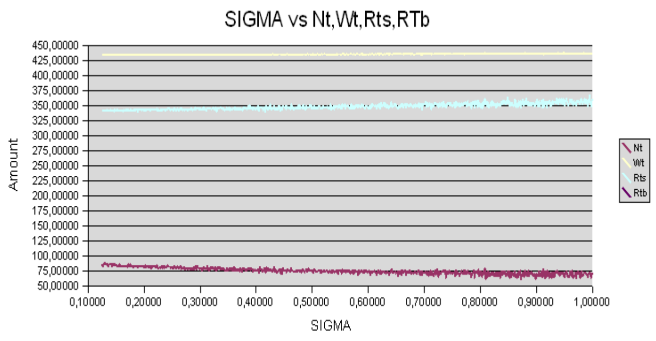


Figure 33

Figure 33 presents the relation between σ and N_t,W_t,R_{ts} and R_{tb}. As σ increase, the number of transactions decrease due to an increase in asks prices (higher variability in generation of asks). Seller and Buyer revenue (R_{ts} and R_{tb} respetively) also increase due to the increase in prices. W_t is N_t times W_{avg} and behaves accordingly.

SHAPE Experiment

The second experiment consisted in the evaluation of the Pareto Index (ς) and its implications on the allocation provided by the auction. As in the previous experiments, we fixed the values of one distribution and we experimented with different values of the other. In this experiment, asks were generated once with a μ fixed at 1.0 and σ fixed to 0.125. $p_{avg} = \sum_{i=1}^{k} \frac{p_{sell_i}}{k}$ was also calculated and fixed as a base price for all the bids. Bids were for a single item and for a single time slot because the aim of the study was to understand how the variability of prices for one resource affects the behaviour of the auction. So we would like to keep the model as simple as possible.

As a result we got that (looking at (2)): $p_{base} = p_{avg}$

Bids have been generated using a *Pareto* distribution with parameters ς > 0 and P_{base} fixed as state before. The generated inverse distribution function was:

$F^{-1}(u) = P_{base}(1-u)^{-1/\varsigma}$ for 0 <= u < 1.

The experiment consisted in 950 executions of the clearing process each one with a different bid input. For each experiment the same 100 asks were used. In contrast, 100 bids were generated at each execution. For bids, we used a value of ς ranging from 0.5 to 10.0, P_{avg} was calculated as the average value of asks. After each execution, ς was increased by 0.01.

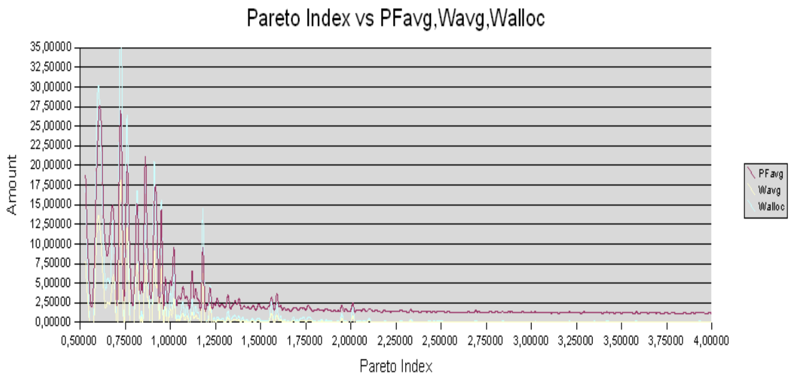


Figure 34

Figure 34 shows the relation of the Pareto Index with the $PF_{avg,}$ W_{avg}. Note that even we computed values ranging form 0,5 and 10.0 Figure 34 only shows the results until ς = 4.0 because the tail of the curve keeps constant until 10.0. Looking at the results we can see that lower values of ς present higher levels of instability leading to high variations of $PF_{avg,}$ and $W_{avg.}$ Close to 1.2 ς stabilizes and keeps almost constant until 10.0.

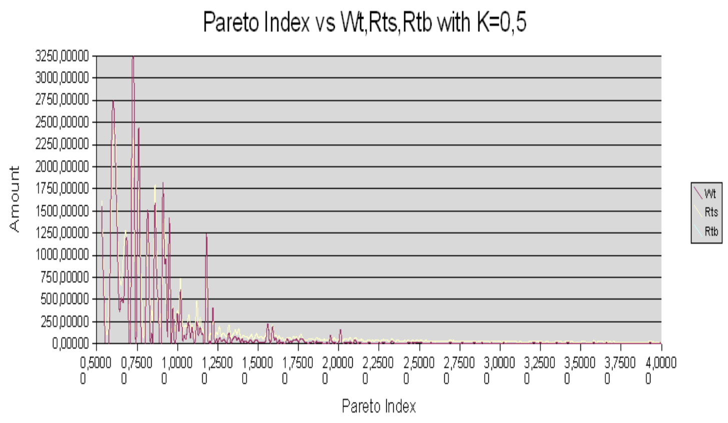


Figure 35

As in the case of the previous experiment, as long as ς increases the number of allocations decreases (see Figure 35). This effect can be attributed to the fact that as ς increases, the proportion of high-value bids is reduced leading to a decrement of the number of bids that are above the price.

K experiment

The next experiment consisted in the evaluation of the pricing policy. As a remainder, the K-pricing policy calculates transaction prices as:

$$P_{final} = k \cdot P_{bid} + (1-k) \cdot P_{sell} \quad (15)$$

K-pricing policy is directly related to how welfare is distributed, leading to theoretically optimal results when *k=0,5*. Note that when *k=0* the mechanism is incentive compatible for sellers whilst when *k=1* it is incentive compatible for buyers.

Our experiment consisted of 100 iterations. Initially *k* was set to 0 and after every iteration the value of *k* was incremented by 0.01 until it reached the value of 1.0.

For each value of k we computed average values of PF_{avg}, W_{avg}, N_t, W_t, R_{ts} and R_{tb} for the execution of an auction ran 1000 times over the same input data.

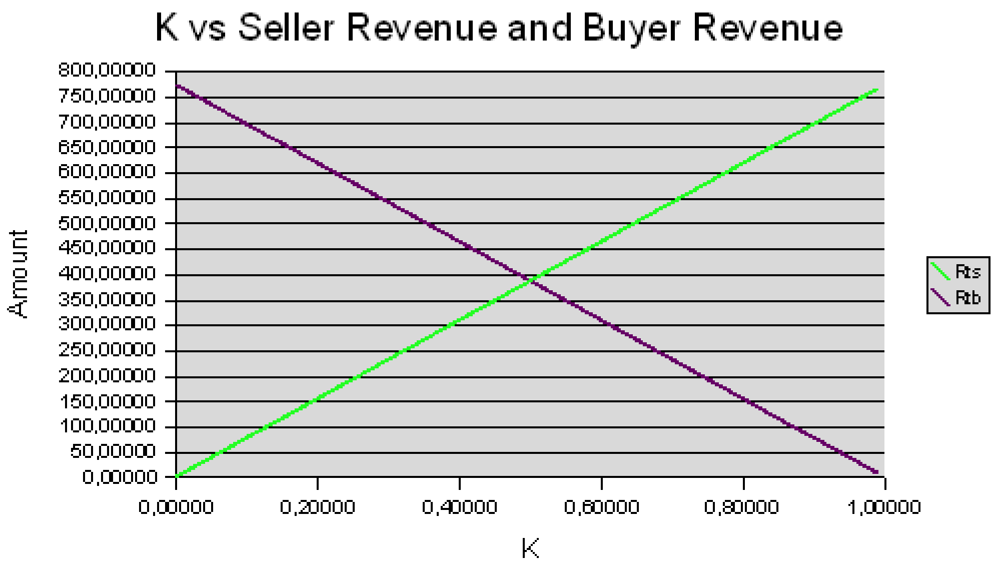


Figure 36 shows how revenue is shared amongst buyers and sellers for different values of *k*.

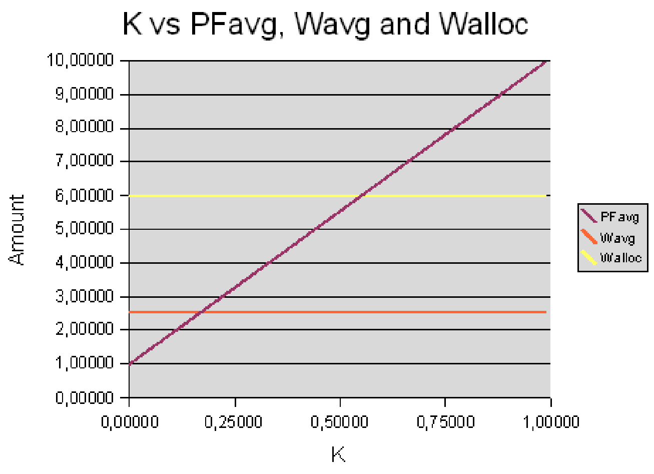


Figure 36

Figure 36 shows how prices increase as long as the value of k gives more weight to bid prices. The welfare of the system keeps constant. The aim of the experiment was to study how the variations of the characteristics of some statistical distribution affect results of the auction. For the case of ask generation we conclude that the variability of the prices does not affect too much the behaviour of the auction, keeping price and social welfare almost constant but at expenses of a decrement of the number of allocations. In the case of bids we conclude that the Pareto Index has a higher impact on the generation of prices. Our experiments showed that generating bids with values of ς ($0.5<\varsigma<1.1$) lead to very unstable behaviour. The main reason for that behaviour can be attributed to the wider variance of generated prices. As long as ς increases the variance of the generated prices is reduced leading to a more stable behaviour. We conclude that using values of ς larger than 1.1 the behaviour of the auction is quite stable apart from a sensible reduction on the number of allocated bids.

Comparison of continuous and discrete double auction

In order to obtain data from the two market institutions, two market instances were activated. The first market instance implemented a continuous double auction (CDA), and the second market instance implemented a scheduled double auction (SDA). Tests were conducted during 500 iterations. In each round, 100 processes trying to buy resources and 100 trying to sell resources were executed. 50 of the buying (selling) processes submitted bids to the CDA and the rest submitted bids to the SDA.

Processes acting as buyers generated bids following a Pareto distribution having established an initial price. Pareto distribution has been considered the most appropriate distribution for generating bids since it has been used to describe the distribution of wealth in the society. Sellers generated their offers following a normal distribution, because of the normal distribution of the costs of the traded items. Prices were updated at end of each round taking into account the variations in supply and demand.

Two important variables were observed, the number of allocations and the social welfare. The first one indicates the efficiency of the mechanism in terms of the number of resources used. Social welfare measures the capacity of the market to allocate the resources to those who need them more. Social welfare has been calculated as the aggregation of the buyer's surplus and seller's surplus.

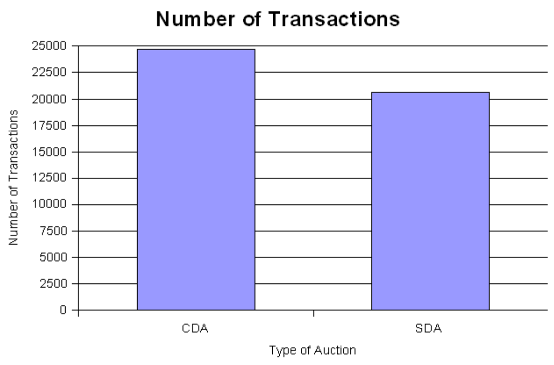


Figure 37

Figure 37 presents the number of successful transactions for both mechanisms. The CDA obtains understandably a higher number of transactions than the SDA. Figure 38 presents the social welfare for the 500 iterations. The CDA obtains a slightly lower aggregated social welfare than the SDA with 4101 less transactions. This is because SDA has the total set of bids/asks whereas CDA is eager. Figure 39 shows the average price evolution in the market as well as the social welfare for each of the iterations. Price evolution is guided by random variables that depend on the levels of activity in the market.

Our tests showed that the social welfare provided by the SDA is higher than the social welfare provided by the CDA. Contrarily, the CDA provides a higher number of allocations which means that a higher number of resources are utilized. The choice of one or the other constitutes a trade-off between economic efficiency and number of allocations. Which mechanism to use is an individual decision that the trader may take based on its strategy.

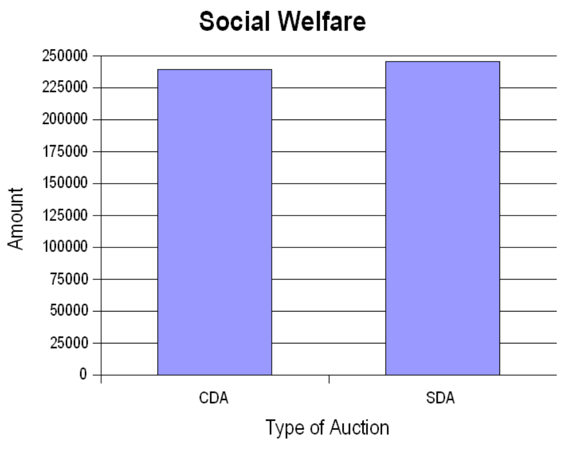


Figure 38

After doing the presented tests, we wondered about improving the social welfare provided by the CDA without decreasing significantly the number of allocations. To do so, we delayed the immediate allocation of the CDA and re-executed the tests. In order to delay the clearing of the CDA it was configured to wait until at least 4 bids had arrived to the market. The results were fairly interesting. As can be seen in Figure 39 and Figure 40, the number of allocations provided by the CDA remained higher than the number of allocations that the SDA provided, contrarily the difference of the aggregated social welfare provided for both methods was reduced nearly a 50%.

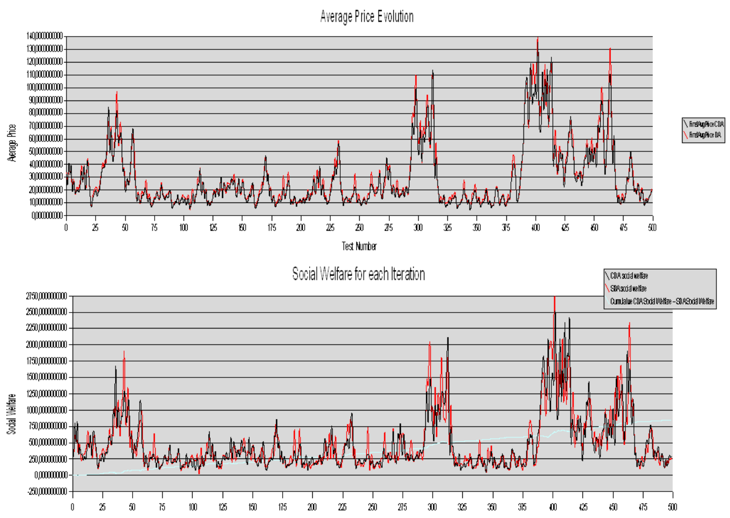


Figure 39

By delaying the clearing of the CDA the social welfare is increased at the expense of reducing number of allocations. Even then, the number of allocations remains higher than that provided by the SDA and the improvement of the social welfare is more significant than the decrement of the number of allocations. A conclusion extracted is that the efficiency of the CDA can be improved by delaying slightly the clearing process. It can be done either by discretizing continuous time into small iterative time frames or by delaying the clearing until a certain number of bids arrive to the market

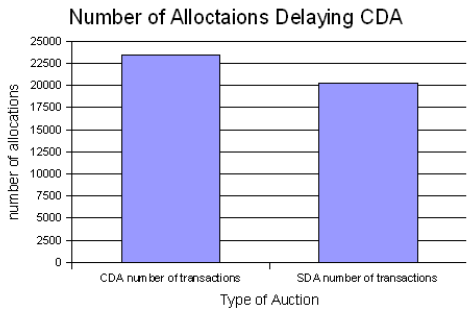


Figure 40

Conclusions

The experiments conducted propose different set of distributions to generate bids. The purpose of the evaluation is to measure the impact of parameters used to generate bid and ask prices on the outcome of the K-Double auction where the outcome is measured in terms of number of transactions (representing allocations), the final price of transactions and the social welfare that is generated.

B.3. Combinatorial auction

Unlike the double auction mechanism, the combinatorial auction permits to trade simultaneously multiple units of multiple resource types. This mechanism is preferable for jobs requiring a combination of resources. The different resources are complementary (the bundle of items is more valuable than the sum of the individual items); it may be worthless for the consumer to obtain a subset of the requested bundle. Combinatorial auctions enable expression of complementarities and allows participants to bid on bundles of items.

In Section 1, we present the design of the combinatorial auction and the scenario requirements. The mathematical formulation of the mechanism is given in Section 2. Finally, in Section 3, we explain how to deal with the presented mathematical problems.

Design of a combinatorial auction

Let us consider a marketplace on which several negotiators (consumers and suppliers) want to trade computer resources. There may be more than one type of resources; in our case, computational and storage resources that are specified by their quantity and quality attributes. Each negociator may request or provide any combinations of these resources.

The aim of the mechanism is to first determine an allocation of resources, and then to provide the trading prices. We say that the mechanism is efficient if it maximizes social welfare where the social welfare is the aggregate of individual utilities obtained by the allocation. We assume that the negotiators submit truthful bids (with their true valuations), the utility is hence computed as the difference of the valuations of winning jobs (or the reservation prices of winning bundles) and the corresponding payments.

The CNSE application as described in D4.7 and D4.2 requires a combination of computational and storage resources. In this scenario, the job (simulation) should complete before a hard deadline (end of class room session). Earlier the job may finish, more the value for the completion, since this leaves sufficient time for discussion within the class-room. Completion of the job beyond the deadline is worthless for the class-room.

The completion time clearly depends on the number, quality and duration of allocated resources. To control the allocation computation time, we would like to minimize the number of bids (XORs) submitted by the consumers. Likewise suppliers must choose the configurations of offered bundles. Feedback from the market may help the negotiators make relevant choices (the selection of optimal resource configurations); information on the time-slots that have heavy demand, the price according to the quality of resources, the supply of different types of resources are essential to help negotiators limit their selections.

We propose to provide this information by performing an iterative combinatorial auction. At each round of the auction, negotiators submit some bids and provisional winners and prices are computed. From this result, consumers and suppliers determine their new bids for the subsequent rounds. Auction terminates when the negotiators do not modify their bids. Here the provisional winners of the last round become the final winners.

Following sections give details of the CA mechanism and the scenario requirements.

Traded commodities

The scenario-specific CA trades two commodities; computation and storage service. The job execution time depends on CPU performance and the level of parallelism defined by number of CPU units. As CPU is worthless without physical memory, the combination is traded as an entity and the RAM specifications is considered as an attribute of the computation service. Storage service is characterized by the number of storage units, storage size and the available throughput which is characterized as the speed of writing on a hard disk (network capacities and transfer time are ignored).

To limit the combinatorial explosion when all possible values are considered, we simply this model by aggregating the quality features of each resource. We propose to consider only realistic and typical configurations for each resource type. The categories of a resource type is ordered such that if a consumer requests a resource of category i, any category j, $j \geq i$ also match. A resource category is considered a commodity since two units of the same category cannot be distinguished. Thus resources are characterized by three attributes: the type of resource (CPU or Storage), the quantity and the quality category.

Time characteristics

The market defines an allocation horizon consisting of discrete time intervals or time slots of fixed and uniform length.

Bid configuration

Agents purchase and offer services by submitting bids to the market platform. Although both supplier and consumer bids contain resource combinations, their structure differ. Supplier bids consist of a set of bundles of which an arbitrary number may be allocated (a bid is an OR ed set of bundles). The sum of all resources in the set of bundles should correspond to the supplier's total trading capacity. The maximum bundle quantity, i.e., the number of OR nodes, is predefined by the market. Bundles are available over a time-frame and give the reservation prices representing minimal acceptable revenue for a bundle per time slot.

Consumers may express their preferences for substitutable resource configurations (jobs), by submitting XOR bids. Only one job should be allocated. A job is specified by the earliest execution start time, the latest end time (corresponds to deadline). Each job gives its valuations[20]. The valuation represents the maximum amount that the bidder is willing to pay.

[20] Bidders are expected to know the valuations.

A job may consist of several conjunctive parts represented by the AND operator. A job can be executed if and only if all its parts are allocated. Each part is defined by a set of resources and the duration for which they are required with a time frame. The consumer may request the parallelization of the job parts (here the durations of all parts are expected to be equal). The time specifications and the parallelization parameter apply to the job and not to the individual parts.

The Allocation characteristics

A valid allocation is one where the all the attributes specified in the job match those of the bundles to which it is allocated; resource attributes, the availability times must match and the aggregate prices of the allocated bundles must not be greater than the valuation of the job. The WDL (Winner Determination Problem) determines an allocation of jobs to bundles that maximizes the social welfare among all valid allocations. We have added some dispersal constraints: a part of a job is allocated to at most one bundle, and for every time-slot, a bundle is allocated to at most one job part. This also implies that at a given time-slot, different parts of a job is allocated to different bundles. Job parts are allocated in contigous time slots(without holes). If parallelization is activated, all job parts are allocated in exactly the same time frame.

The pricing characteristics

Payments of winning bids are computed after the determination of optimal allocations. Even though desirable properties for pricing are *budget-balance*, *individual rationality*[21] and *incentive compatibility*[22], a well-known result[23] states that no exchange can be allocative-efficient, budget-balanced and individually rational at the same time[24]. So one must choose to favour some properties to the detriment of the others; we choose to enforce the budget-balance and individual-rationality properties, since the market must not be deficient nor dependant on subsidies, and we want the negotiators to be volunteer for participating.

There are two pricing options: ***bundle pricing*** which consists in determining prices of winning bundles, and the ***item pricing*** which consists in determining prices of individual items. Bundle pricing permits to express the complementarities between the items, but item pricing gives more interpretable information about the market. This facilitates the decision making for negotiators. Consumers can restrict their choice of resource configurations to those whose aggregated price computed by using the item prices do not exceed the valuations.

This pricing must be budget-balanced and individually rational. One desirable property of item prices is its closeness to *market clearing* prices: the loosers find out that their bid is too low (for the consumers) or that their reservation price is to high (for the providers). However such prices cannot be guaranteed to exist in combinatorial auctions. We may need to find approximates that minimize the maximum gap between the bids (resp. the reservation prices) and the computed prices of the bundles in case the market clearing property is not verified. Also, in order to ensure the individual-rationality property, we may have to introduce a gap between the prices paid for winning bundles and the sum of the prices of the items that make it up. Again we try to minimize the extent of this gap.

Mathematical formulation

Notation

[21] All agents have positive expected utility from the participation.
[22] The best strategy for the negotiators is to report truthfully their valuations.
[23] Satterthwaite
[24] Independently of incentive compatibility

Let R be the set of resources and A_r the set of quality categories of resource r. The length of the allocation horizon is defined by $T = \{1,...,\bar{t}\}$.Let M be the set of suppliers. For each supplier $m \in M$, B^m denotes the set of bundles he offers (the number of bundle in B^m is upper-bounded by a predefinite quantity $\bar{b}$). Each bundle b of supplier m is defined by four parameters :

- the earliest available start time slot denoted by $c^{mb} \in T$;
- the latest available end time slot denoted by $f^{mb} \in T$;
- the reservation price per time slot denoted by p^{mb};
- the resource attributes denoted by d^{mb}_{ri}, $r \in R$, $i \in \{0,1\}$ (the value $d^{mb}_{r,0}$ corresponds to the quantity of resource r offered by supplier m in the bundle b, and $d^{mb}_{r,1} \in A_r\}$ specifies the quality of the resource).

Let N be the set of consumers. The consumer $n \in N$ bid consists of a set of jobs J_n. A job j of consumer n is specified by:

- the earliest execution start time slot denoted by $e^{nj} \in T$;
- the latest execution end time slot denoted by $l^{nj} \in T$;
- the valuation for the job denoted by v^{nj} ;
- a set of job parts denoted by K^{nj} ;
- the parallelization of job parts denoted by $\gamma^{nj} \in \{0,1\}$ (γ^{nj}=1 if the parallelization is activated, 0 otherwise);
- for each part $k \in K^{nj}$ of the job :
    - the resource attributes denoted by a^{njk}_{ri}, $r \in R$, $i \in \{0,1\}$ (the value $a^{njk}_{r,0}$ corresponds to the quantity of resources r required by consumer n for the part k of job j and $a^{njk}_{r,1}$ specifies the quality requirements);
    - the duration $q^{njk} \in \{1,...,l^{nj} - e^{nj} + 1\}$ (the number of required time slots).

The Winner Determination Problem

To state the winner determination problem as an integer linear program, we need first to define two groups of decision variables:

- $z^{nj} \in \{0,1\}$, with $n \in N$ and $j \in J_n$, specifies if job j of consumer n is allocated or not.
- $y^{njk}_{mb,t} \in \{0,1\}$, with $n \in N$, $j \in J_n$, $k \in K^{nj}$, $m \in M$, $b \in B^m$ and $t \in T$, indicates wether job part k of job j of consumer n is allocated to bundle b of supplier m with time slot t as start time.

The WDP can now be formulated as the following integer linear program:

$$MaxV = \sum_{n\in N}\sum_{j\in J_n} z^{nj}v^{nj} - \sum_{n\in N}\sum_{j\in J_n}\sum_{k\in K^{nj}}\sum_{m\in M}\sum_{b\in B^m}\sum_{t\in T} y^{njk}_{mb,t}p^{mb}q^{njk}$$

under the constraints :

$$\forall n\in N, \sum_{j\in J_n} z^{nj} \le 1$$
$$\forall n\in N, \forall j\in J^n, \forall k\in K^{nj} \sum_{m\in M}\sum_{b\in B^m}\sum_{t\in T} y^{njk}_{mb,t} = z^{nj}$$
$$\forall m\in M, \forall b\in B^m, \forall t\in [c^{mb}, f^{mb}], \sum_{n\in N}\sum_{j\in J_n}\sum_{k\in K^{nj}}\sum_{t'=t-q^{njk}+1}^{t} y^{njk}_{mb,t'} \le 1$$
$$\forall n\in N, \forall j\in J^n, \forall t\in [e^{nj}, l^{nj}], \forall k\ne k'\in K^{nj},$$
$$\gamma^{nj}\sum_{m\in M}\sum_{b\in B^m} y^{njk}_{mb,t} = \gamma^{nj}\sum_{m\in M}\sum_{b\in B^m} y^{njk'}_{mb,t}$$
$$\forall n\in N, \forall j\in J^n, \forall k\in K^{nj}, \forall m\in M, \forall b\in B^m,$$
$$\forall t\in T\setminus[\max(e^{nj}, c^{mb}), \min(l^{nj}, f^{mb} - q^{njk} + 1)], y^{njk}_{mb,t} = 0$$
$$\forall n\in N, \forall j\in J^n, \forall k\in K^{nj}, \forall m\in M, \forall b\in B^m,$$
$$\forall t\in T, \forall r\in R, \forall i\in A_r such that a^{njk}_{ri} \le d^{mb}_{ri}, y^{njk}_{mb,t} = 0$$
$$\forall n\in N, \forall j\in J^n, \forall k\in K^{nj}, \forall m\in M, \forall b\in B^m, \forall t\in T, y^{njk}_{mb,t} \in \{0,1\}$$
$$\forall n\in N, \forall j\in J^n, z^{nj}\in\{0,1\}$$

The objective function aims at maximizing the social welfare of participants. This is represented by the difference of the sum of job valuations for allocated jobs and the sum of reservation prices for the associated bundles and time slots. Constraints (1) ensure the representation substitute in job allocation (XOR). Constraints (2) force variables y and z to be coherent. This guarantees also that a job part cannot be allocated by more than one bundle. Constraints (3) guarantee avoidance of overlapping jobs. Constraints (4) ensure the allocation of job parts of a given job in exactly the same time slots when the parallelization is activated. Constraints (5) expresses the matching of availability time frames. Constraints (6) check quantity and quality attributes. Finally constraints (7) and (8) ensures that variables y and z are binary.

Pricing scheme

Let us assume that the WDP has been solved with $\bar{y}$ and $\bar{z}$ as optimal solution. Let M_W and N_W denote the subsets of winning suppliers and consumers. For $m\in M_W$, B^m_W denotes the subset of B_m containing the allocated bundles. Likewise, for $n\in N_W$), J^n_W denotes the subset of J_n containing the allocated job ($|J^n_W|=1$). We consider also the subsets $B^m_L = B^m \setminus B^m_W$ and $J^n_L = J^n \setminus J^n_W$ of loosing bundles and jobs. Let finally δ^{mb}, $m\in M_W$, $b\in B^m_W$, denote the total number of time slots allocated for bundle b of supplier m. Let us consider the following variables:

- $\mu^{mb}\in \mathrm{R}^+$, with $m\in M_W$ and $b\in B^m_W$, is the price to be received by the supplier m for its allocated bundle b;

- $\nu^{nj} \in \mathrm{R}^{+}$, with $n \in N_W$ and $j \in J_W^n$, is the price to be payed by the consumer n for its winning job j;

- $\Pi^{r\alpha} \in \mathrm{R}^{+}$, with $r \in R$ and $\alpha \in A_r$, denotes the price of one unit of resource r of category α per time slot.

We consider also four families of slack variables denoted by :

- $\varepsilon_1^{mb} \in \mathrm{R}$, $\forall m \in M_W$ and $\forall b \in B_W^m$;
- $\varepsilon_2^{nj} \in \mathrm{R}$, $\forall n \in N_W$ and $\forall j \in J_W^n$;
- $\varepsilon_3^{mb} \in \mathrm{R}^{+}$, $\forall m \in M$ and $\forall b \in B_L^m$;
- $\varepsilon_4^{nj} \in \mathrm{R}^{+}$, $\forall n \in N_L$ and $\forall j \in J^n$.

The the pricing comes down to the resolution of the following linear program:

$$\min_{\pi,\mu,\nu,\varepsilon,Z} Z$$

under the constraints :

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

$$\forall m \in M_W, \forall b \in B_W^m, \mu^{mb} = \delta^{mb} \sum_{r \in R} d_{r,0}^{mb} \Pi^{r,d_{r,1}^{mb}} + \varepsilon_1^{mb}$$

$$\forall n \in N_W, \forall j \in J_W^n, \nu^{nj} = \sum_{k \in K^{nj}} q^{njk} \sum_{r \in R} a_{r,0}^{nj} \Pi^{r,a_{r,1}^{nj}} + \varepsilon_2^{nj}$$

$$\forall m \in M, \forall b \in B_L^m, p^{mb} \geq \sum_{r \in R} d_{r,0}^{mb} \Pi^{r,d_{r,1}^{mb}} - \varepsilon_3^{mb}$$

$$\forall n \in N_L, \forall j \in J^n, v^{nj} \leq \sum_{k \in K^{nj}} q^{njk} \sum_{r \in R} a_{r,0}^{nj} \Pi^{r,a_{r,1}^{nj}} + \varepsilon_4^{nj}$$

$$\forall m \in M_W, \forall b \in B_W^m, |\varepsilon_1^{mb}| \leq Z$$

$$\forall n \in N_W, \forall j \in J_W^n, |\varepsilon_2^{nj}| \leq Z$$

$$\forall m \in M, \forall b \in B_L^m, 0 \leq \varepsilon_3^{mb} \leq Z$$

$$\forall n \in N_L, \forall j \in J^n, 0 \leq \varepsilon_4^{nj} \leq Z$$

$$\forall m \in M_W, \forall b \in B_W^m, \mu^{mb} \geq p^{mb} \delta^{mb}$$

$$\forall n \in N_W, \forall j \in J_W^n, \nu^{nj} \leq v^{nj}$$

$$\forall r \in R, \forall \alpha \in A_r, \Pi^{r\alpha} \geq 0$$

The objective of this program is to minimize the value of Z which corresponds to the maximum gap ε. Constraint (9) ensures the budget balance of the market. Constraints (10) and (11) aim at minimizing the gaps ε_1 and ε_2 between the prices paid and received by winning negotiators and the sum of the individual prices of the corresponding resources. Constraints (12) and (13) ensures that the market clearing property is not too much violated: if possible the loosing bundles have reservation prices greater than the sum of the individual prices of the corresponding resources and the loosing consumers have valuations lower than the market prices of resources; otherwise the gaps ε_3 and ε_4 are minimized. Constraints (14), (15), (16) and (17) define the value of Z as an upper-bound for all ε. Constraints (18) guarantee that the prices received for allocated bundles are greater than their reservation prices. Constraints (19) guarantee that the prices paid by consumers are lower than the valuations of the winning jobs. Constraints (20) ensures that the computed prices are nonnegative.

Solving the combinatorial auction

The winner determination problem is formulated as an integer linear program. Actually, this problem comes down to a scheduling problem. In this type of problems, the decision maker must find a way to successfully manage resources in order to execute jobs (or produce products) in the most efficient way possible. He needs to design a schedule which satisfies the jobs requirements and which optimize some objective function such as for instance minimizing the makespan, or minimizing the amount of consumed ressources. So the only difference between the WDP and a scheduling problem lies in the objective function.

Such problems have been studied a lot in the last decades by the combinatorial optimization community. The WDP is classified among the most difficult problems (it is NP-hard). In practice, it results in the impossibility to solve optimally this problem when the size of the data is large, although some sophisticated and powerful solvers have been developped to deal with such difficult problems.

We therefore propose two methods to solve the WDP. If the size of the data is small enough, it is possible to use a solver in order to find the optimal solution. This can nevertheless take a certain amount of time, even for instances of reasonable size. If the size of the problem is too large or if the negotiator or the service which initiates the auction specifies a maximum time of completion, then it can be necessary to use a heuristic (an approximate method) that provides fastly an allocation satisfying the constraints of the problem and with a good value of the social welfare (if possible close to the optimal one).

The pricing can be formulated as a linear program with continuous variables. Thereby it can be solved quite easily by applying the simplex algorithm.

Conclusion

We have developed a combinatorial auction specifically for Grid4All trading expectations. The Winner Determination Problem has been implemented and evaluated using a set of generated instance data representing jobs and bundles. The detailed report [CAS-7] is provided along with this deliverable. We have evaluated the performance of the auction through four numerical experiments to understand the influence of the different quantity and quality parameters. The model scales acceptably well with increase in the number of time-slots and number of agents, until a limit of 500 agents.

Our next steps are to implement the pricing algorithm. The first implementation will focus on resolution of the linear program using simplex algorithms. We will evaluate the results before turning to designing heuristic algorithms.

B.4. Distributed Market Information System

Main Interactions

A query is issued by a trader to obtain information on the resource markets as illustrated in Figure 5. The trader creates filters to narrow the search for information, and a handler for the query results. These objects are passed to DMIS, which executes a query using its routing layer. The routing process queries nodes on the overlay using the filter and aggregates received information. The current prototype provides the following queries on compute nodes. The parameters of queries are the QueryHandler, the number of units of the resource, the start and end times within which the information is aggregated.

Name	Description	Comment
getAveragePrice	Average price for one or more units of resource, per unit-time, and between specific start and end times of the day.	This will affect the value in the bid, the starting time and the ending time between which the resources are required. This is not independent of the utility function of the application itself.
getVolume	Total volume of trade of a resource, between a specific start and end time of a certain day.	The volume of traded products helps to find good market sectors and for statistics of the resource providers.
getMinimumPrice	Minimum price for one or more units of resource at specific times in the day, for a unit of time.	This information is needed for bidding strategies such as Zero Intelligence Plus (ZIP)
getMaximumPrice	Same as above, but maximum.	This information is needed for bidding strategies such as Zero Intelligence Plus (ZIP)
getAverageClearingTime	Average time to clear auctions.	To decide the time at which the negotiator should start its negotiation taking into account the time at which resources are needed.
getTotalDemand (Offer)	Total demand (offer) for a specific resource	This may be more useful for providers. But even for consumers, more the demand at specific times of day, the higher the price is likely to be at those times. Hence this may direct the consumers to set the best time specifications.

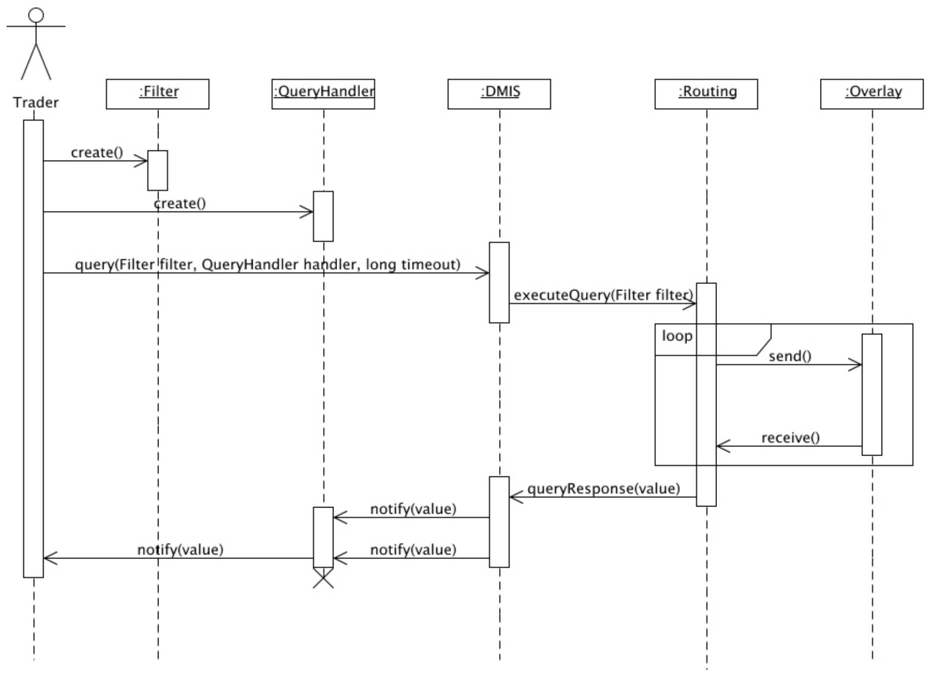


Figure 41 Example internal query process

Figure 41 shows a trader that subscribes to a topic/content and another trader publishing an event, which will notify the subscriber. The subscriber creates a filter that defines the interested events. The created subscriptions are installed by the DMIS in its routing layer. Subscribers are notified when a new event matches a set filter. Similarly publishers on arrival of new events, e.g. establishment of agreement between buyer and seller, send the event to the DMIS which in turn forwards to the routing layer. Routing layers notify their DMIS on a new incoming event. The DMIS then notifies registered *SubscriptionHandler*.

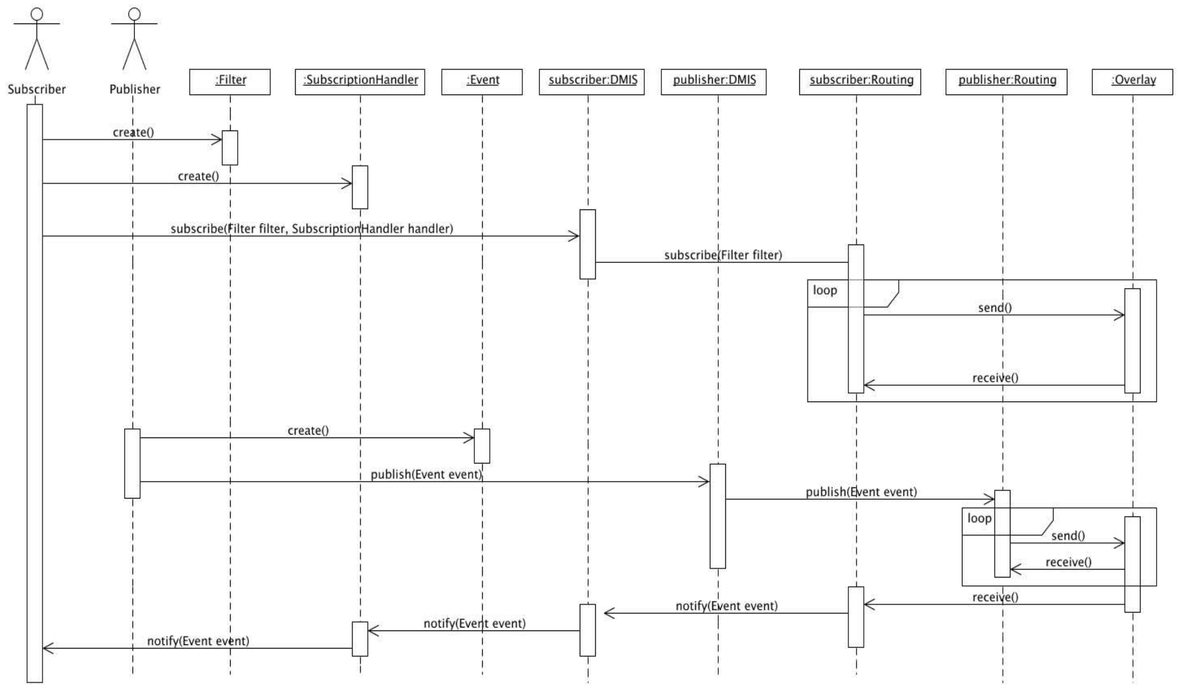


Figure 42 Internal publish and subscribe process

Uncertainty Management

The self-management algorithm for the uncertainty calculates the size of the simple data in for of hops (size of sample data is approximately 2^h within a binary tree). For example, the users define the Confidence Interval (CI) with 95\% or 99\%. The t-table provides the value of $t_{(n,CI)}$. As result, the self-management components adapt its approximations within to keep the results in the tolerance interval, defined by the user.

```
Algorithm 1: Uncertainty self-management algorithm.
  Input: Confidence Interval (CI)
  Input: $\alpha$
  Input: hops
  Input: $t_{(n,CI)}$
  Input: $P_1..P_n$
  Output: hops
  forall prices of $P_1..P_n$ do
    add price to sum;
  end
  median = $sum/countresults$ ;
  forall prices of $P_1..P_n$ do
    add $(median - price)^2$ to $sum_variance$;
  end
  $\sigma$= $sqrt$ from $sum_variance/count_results$;
  $\mu = t_{(n,CI)} * \sigma * 100/\sqrt{n} * median$;
  if $\mu > \alpha$ then
    increment hops by 1;
  else
    decrement hops by 1;
  end
```

Figure 43

Efficient routing for information acquisition

The B-Tree based routing algorithms prototyped within the DMIS and the results of preliminary evaluations are presented in the accompanying technical report [MIS-7].

B.5. Currency Management System

The **CMS Bank Service Layer** provides operations to perform account creation and deletion as well as modifications to these accounts when performing, for example, a transfer of funds. It relies on the guarantees supported by the lower layer (Transactional Data Layer) to ensure the ACID properties of its operations and to reliably store user accounts. This layer is also responsible to define and implement regulation policies regarding the amount of currency that a user account may hold at any time, the maximum amount per transactions, etc.

The **Transactional Data Layer** provides mechanisms to perform ACID transactions when modifying objects stored in the lower layer. To provide those semantics, the data will be accessed in mutual exclusion to avoid transaction inconsistencies. It relies on the probabilistic guarantees supported by the lower layer (Mutable Consistent Data Layer) to reliably store and retrieve such objects.

The **Mutable Consistent Data Layer** provides an enhanced DHT interface to support the *update* operation as well. Moreover, it is responsible to deliver the most up to date data stored within the system. This Layer will be based on the DHT API provided by the Niche peer-to-peer middleware.

The **KBR Layer:** KBR stands for *Key Based Routing*. As its name suggests its responsibility is to provide mechanisms to communicate different nodes based on their key interval responsibility. We will use the KBR Layer provided by the DKS middleware without any modification. As long as DKS uses a standard KBR API, this layer would be replaced by any other middleware providing this kind of routing mechanisms.

This section presents the main interfaces and programming APIs to the CMS.

```
/**
 * The <code>CMSInterface</code> class
 *
 * API offered by CMS to the Agreement Manager/Buyer Agent. This API will be
 * the interface offered once CMS is wrapped as a single Fractal Component.
 * @author Xavier León
 *
 */
public interface CMSInterface {

      /**
       * Open an account for a given user identified by its credentials within
       * the CMS infrastructure.
       * @param user Credentials of the user opening the account. An user can
       * not have more than one account associated with each Credentials.
       * @return An AccountID which identifies uniquely the account created
       * for the user.
       * @throws AccountAlreadyExistsException
       * @throws InvalidCredentialsException
       */
      public AccountID openAccount(Credentials user)
                        throws   AccountAlreadyExistsException,
                                 InvalidCredentialsException;


      /**
 * Closes the account identified by an AccountID. Users allowed to
 * close it are the owner of the account -- who openend it -- or the
 * admin of CMS otherwise.
 * @param accountID AccountID of the account to be closed
 * @param owner Credentials of the user which is performing the
 * operation.
 * @return A Boolean representing wether the operation has finished
 *        correctly.
 * @throws AccountNotFoundException
 * @throws InvalidCredentialsException
 */
public Boolean closeAccount(AccountID accountID, Credentials owner)
                throws    AccountNotFoundException,
                          InvalidCredentialsException;

/**
 * Query the Account information of the user.
 * @param accountID AccountID of the account to be queried.
 * @param owner Credentials of the user which is performing the
 * operation.
 * @return The AccountInfo matching the AccountID provided by the user.
 * @throws AccountNotFoundException
 * @throws InvalidCredentialsException
 */
public AccountInfo queryAccount(AccountID accountID, Credentials owner)
                throws    AccountNotFoundException,
                          InvalidCredentialsException;

/**
 * Transfer funds from one account to another for a given amount of
 * currency. The only user allowed to perform this operation is the
 * owner the source account.
 * @param src AccountID of the source account (Buyer account).
 * @param dst AcocuntID of the destination account (Seller account).
 * @param amount Amount of currency to be transfered.
 * @param buyer Credentials of the buyer agent which performs
 *       the transaction.
 * @return A ticket representing the proof-of-payment.
 * @throws AccountNotFoundException
 * @throws InvalidCredentialsException
 * @throws NotEnoughFundsException
 * @see edu.upc.cnds.cms.api.TransferReceipt
 */
public TransferReceipt transferFunds(AccountID src,
                                     AccountID dst,
                                     Double amount,
                                     Credentials buyer)
                throws    AccountNotFoundException,
                          InvalidCredentialsException,
                          NotEnoughFundsException;
```

```
/**
 * Increase the user account associated to the AccountID with the given
 * amount of currency. This method will be only accesed by system
 * administrators to deposit funds due to any reason (e.g. to deposit
 * initial funds to the account, user wins a dispute against a provider,
 * etc).
 * @param accountID AccountID of the account balance to be increased.
 * @param amount Amount of currency to be deposit within the account.
 * @param admin Credentials of the admin user of CMS.
 * @return A Boolean representing wether the operation has finished
 *        correctly.
 * @throws AccountNotFoundException
 * @throws InvalidCredentialsException
 */
public Boolean depositFunds(AccountID accountID,
                                      Double amount,
                                      Credentials admin)
                   throws   AccountNotFoundException,
                            InvalidCredentialsException;
```

```
**
	 * Decrease the user account associated to the AccountID with the given
	 * amount of currency. This method will be only accesed by sysadmins to
	 * withdraw funds due to any reason (i.e. user loses a dispute against a
	 * provider).
	 * @param accountID AccountID of the account balance to be decreased.
	 * @param amount Amount of currency to be withdrawn.
	 * @param admin Credentials of the admin user of CMS.
	 * @return A Boolean representing wether the operation has finished
	 *         correctly.
	 * @throws AccountNotFoundException
	 * @throws InvalidCredentialsException
	 * @throws NotEnoughFundsException
	 */
	public Boolean withdrawFunds(AccountID accountID,
	                             Double amount,
	                             Credentials admin)
	                throws AccountNotFoundException,
	                       InvalidCredentialsException,
	                       NotEnoughFundsException;
```

C.Scheduling service

The papers presenting the main research results [SS-1] [SS-2] [SS-3] and [SS-4] are presented as a separate document along with this deliverable.

The following Figure 44 describes the scheduler interface and the relevant input parameters. The *Resource* object represents a computational resource. The CPU speed and network speed are restricted to three value categories. The times at which the resource is available is also represented for purposes of establishing the planning.

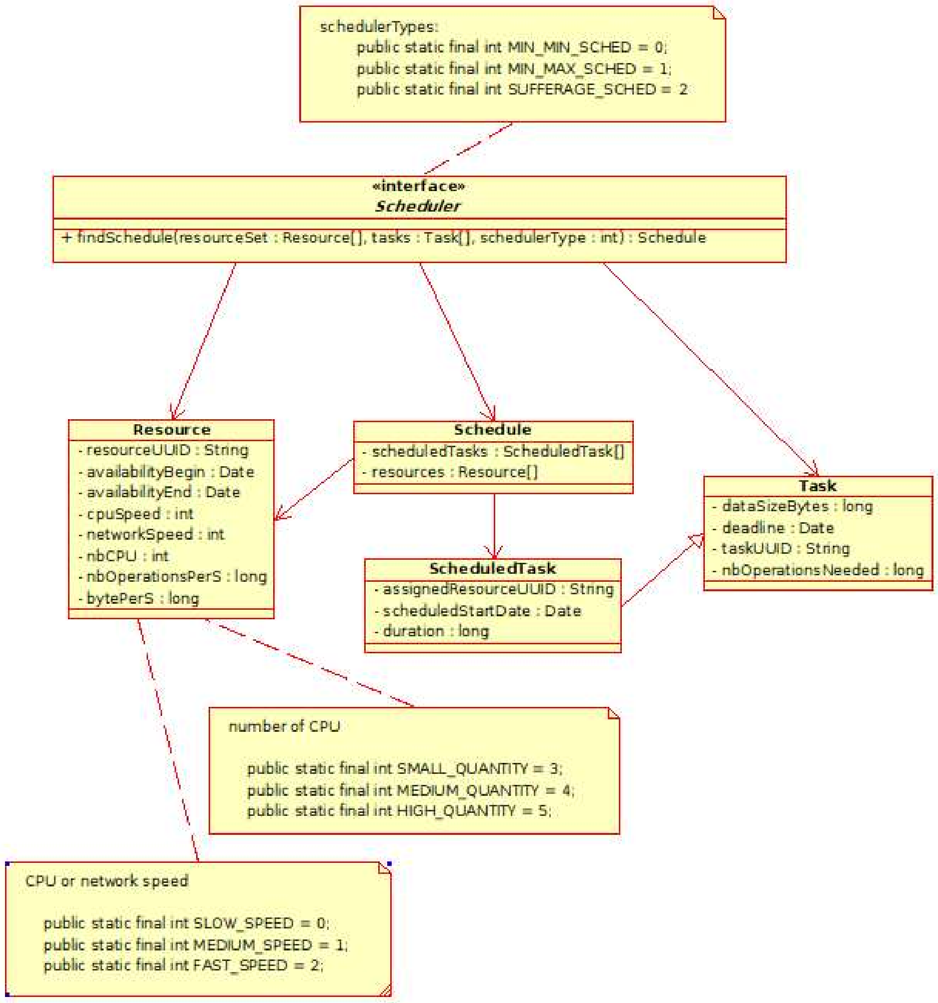

schedulerTypes:
public static final int MIN_MIN_SCHED = 0;
public static final int MIN_MAX_SCHED = 1;
public static final int SUFFERAGE_SCHED = 2
«interface»
Scheduler
+ findSchedule(resourceSet : Resource[], tasks : Task[], schedulerType : int) : Schedule
Resource
- resourceUUID : String
- availabilityBegin : Date
- availabilityEnd : Date
- cpuSpeed : int
- networkSpeed : int
- nbCPU : int
- nbOperationsPerS : long
- bytePerS : long
Schedule
- scheduledTasks : ScheduledTask[]
- resources : Resource[]
ScheduledTask
- assignedResourceUUID : String
- scheduledStartDate : Date
- duration : long
Task
- dataSizeBytes : long
- deadline : Date
- taskUUID : String
- nbOperationsNeeded : long
number of CPU
public static final int SMALL_QUANTITY = 3;
public static final int MEDIUM_QUANTITY = 4;
public static final int HIGH_QUANTITY = 5;
CPU or network speed
public static final int SLOW_SPEED = 0;
public static final int MEDIUM_SPEED = 1;
public static final int FAST_SPEED = 2;

Figure 44 Scheduler interface

The [Tableau 3 MinMin heuristic] describes the default MinMin algorithm implemented by the Scheduling service.

```
tasks are defined by their size
resources are defined by their speed, their begin of availability and their end of availability
schedule (tasks, resources):
---
 sort resources ascending by avail_begin
 WHILE we have resources AND we have unassigned tasks DO
  assign the current task to the first resource in the resources list
   which have a big enough (avail_end - avail_begin) regarding
   the task's size
  IF we were not able to assign the task THEN
   remove it from the tasks list
  ELSE
   update affected resource's avail_begin date
   IF avail_begin == avail_end THEN /** rare **/
    remove resource from resources list
   ELSE
    sort resources ascending by avail_begin by only moving the
     last affected resource
   END IF
  END IF
 END WHILE
```

Tableau 3 MinMin heuristic

INT = Internal, only for members of the Consortium (excluding the EC services).
This level typically applies to internal working documents, meeting minutes etc., and cannot be used for contractual project deliverables.

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