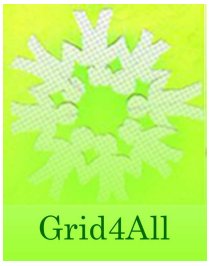


Project no. 034567

Grid4All

Specific Targeted Research Project (STREP)

Thematic Priority 2: Information Society Technologies

D2.1 Requirements for Grid4All Virtual Organisations and Resource Management and State of the art analysis

Due date of deliverable: 1st June 2007

Actual submission date: 28th June 2007

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

Organisation name of lead contractor for this deliverable: FT, INRIA, UPC, UPRC

Contributors: Fabienne Boyer, Jakub Kornas, Jean-Bernard Stefani, Nikos Parlavanzas, Noel de Palma (INRIA), Adam Ouorou, Eric Gourdin, Nejla Amara, Ruby Krishnaswamy (FT), Leandro Navarro, Rene Brunner, Xavier Leon, Xavier Vilajosana (UPC), Derrick Kondo, Gilles Fedak, Paul Malecot (INRIA), Alexandros Valarakos, Andreas Papasalouros, George Vouros, Konstantinos Kotis, Symeon Retalis (UPRC), Jorge Quiane-Ruiz, Patrick Valduriez, Philippe Lamarre (INRIA)

Release 1

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

Table of Contents

Table of figures

Grid4All list of participants

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

1. Executive Summary

This document is part of a research project partially funded by the IST programme of the Commission of the European Communities as project number IST-FP6-034567. This report is a public version of the document "D2.1 – Requirements for Grid4All Virtual Organisations and Resource Management and State of art analysis", as specified in the Grid4All Annex 2 "Description of Work". The objective of Grid4All is to provide middleware support and higher level services for the creation and maintenance of virtual organisations formed of autonomous entities in an open environment. This aim of this project is to address non-conventional Grid users, that is, not just large enterprises and scientific institutions, but small organisations, and individual users on the Internet. This report corresponding to Deliverable report D2.1 presents the requirements, detailed state of art analysis, and architectures for both the management middleware and higher level services required to support maintenance and evolution of virtual organisations. This report is structured in four major parts. Open and volatile environments require middleware with self-management and self-organisation capabilities. The first chapter presents an architecture based framework that takes its source from autonomic computing and addresses essentially creation, deployment, and maintenance of virtual organisations. The second chapter addresses allocation of computational resources with an approach based on open resource markets. Information and discovery services are essential for the operation of virtual organisations. The third chapter addresses the heterogeneity through an approach based on semantics. Virtual Organisations are created to satisfy specific business and technical objectives. The final and fourth chapter presents one usage of virtual organisations that targets domestic users. The technical objective is the development of a scheduling service appropriate to schedule embarrassingly parallel applications such as video transcoders, using resources on the Internet.

2. Autonomic VO management framework – requirements, state of art, and architecture

2.1 Introduction

In their “Anatomy of the Grid” article [18], Foster and al. defined the “real and specific problem that underlies the Grid concept” as the “coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations”. “Resource sharing” in this paper refers to access by multiple users to all manners of processing, storage, and communication resources, as well as software, data, and other resources required by collaborative problem solving in different domains. Also according to this paper, a *virtual organization* (VO) can be understood as a set of individuals and/or institutions that pool resources and collaborate in order to achieve a common goal. A virtual organization defines and manages the resources available for the participants, the rules for accessing and using the resources, and the conditions under which the resources can be used. This view of the Grid concept, which has potentially strong impacts on Grid architectures (for instance, by emphasizing the management of brokering relationships and rules between VO participants and contributed resources), has been widely adopted by the Grid community (see for instance a recent survey on Data Grids [53]). Furthermore, it resonates nicely with notions of virtual organization developed by the information management systems community [28], where the emphasis is placed on the management of requests for services (e.g. information, advice, transactions), and their satisfaction through available services (e.g. databases, expert systems, human experts).

This notion of virtual organization is at the core of the Grid4All project, whose goal is in part to provide automated support for the creation and operation of virtual organizations, involving individuals and different types of organizations (including small and non-commercial ones such as schools, local councils, or families), and the pooling of resources from these potentially very diverse origins. This goal raises a number of issues concerning the management of dynamic virtual organizations. A discussion of these issues can be found e.g. in [41, 42]. We can summarize them as two broad questions:

- How is a virtual organization established/terminated and configured, including the identification of pooled resources and participating organizations, the definition of its objectives and policies, and the set up of its operational activities?
- What management functions are required to support the operation of a virtual organization, including operations for dealing with memberships, access control and authorization, resource allocation and management, monitoring and control of operational activities, negotiation and provisioning of Service Level Agreements (SLA), dynamic federation of multiple virtual organizations?

These questions need to be answered in the context of general requirements pertaining to the relationships between virtual organizations and their participants. For instance, from [18, 44, 56], we can identify the following:

- Users may be members of one or more virtual organizations. A resource can be used in one or more virtual organizations.
- Users may have several roles within a given virtual organization. Virtual organization policies may constrain access to a resource and operations that can be carried out on a resource, based on user identities or user roles.
- It should be possible to list resources and operations to which a virtual organization member or role has access

In the context of the Grid4All project, target application scenarios imply non-professional users in positions of virtual organization administrators. This, coupled with the heterogeneity, dynamism and varying scale of the target environments, calls for an “autonomic” approach [26] to virtual organization management where the presence of humans in the system management loop is minimized (ideally limited to the statement and supervision of high-level goals and policies). We summarize this under a third broad question:

- How may management functions of a virtual organization automated, to minimize the intervention of human administrators, and to allow self-configuration, self-healing, and self-optimization on the basis of high-level virtual organization management goals and policies?

We describe in this document the first elements of an architectural and software engineering framework for the construction and management of virtual organizations. The framework is intended as a first step towards the systematic handling of the above issues, adopting what we describe below as a *control-based* and *architecture-based* approach to distributed system and dynamic virtual organization management. The key elements of our approach to virtual organization management, detailed below, are the following:

- *Domain structure* Virtual organizations are identified with management domains (in the sense of domain and policy-based system management, see e.g. [50]), which can be hierarchically organized and federated. Each virtual organization encompasses an overlay network connecting participating nodes and resources.
- *Architecture basis* Managed resources, as well as management functions, are modelled and/or implemented as components with explicit dependencies, according to software architecture concepts (as presented and motivated, e.g., in [47]).
- *Control basis* Management functions are conceived as distributed feedback control loops (following e.g. the general outline in [13]), organized according to the so-called MAPE-K (Monitor, Analyze, Plan, Execute – Knowledge)[1] model [26].

The document is organized as follows. Section 2.2 reviews related work. Section 2.3 discusses the main elements of our approach, including a view of virtual organization as autonomic systems, and the notion of architecture-based management. Section 2.3 presents the main elements of our proposed framework for virtual organization management. Section 2.4 focuses on the design and implementation of a key management service in or framework: the deployment service. Section 2.6 concludes the document.

2.2 Related work

The development of a framework for virtual organization management in a highly distributed environment relates to many different areas of research. We single out (and discuss relevant works in) two broad areas which are directly relevant to the work reported in this document and to Grid4All objectives: (1) Grid infrastructure services, and in particular virtual organization management services; (2) Autonomic systems.

Concerning Grid infrastructure the reference specification is the Open Grid Services Architecture (OGSA) [19], and the reference implementation is the Globus Toolkit (see e.g. [17] for a recent overview, and references to more detailed descriptions of specific capabilities). Although the OGSA reference architecture mentions self-management as capabilities expected from an open grid infrastructure, the supporting Globus toolkit mostly provides a set of specific capabilities with some elements of automated management operation (e.g. with resource management and execution management services, and, more recently, for dynamic deployment [38]). Several elements of the Globus toolkit can however be exploited in our framework for the provision of so-called management services, i.e. functions required in the implementation of VO-wide automated management functions. For instance, monitoring and discovery services in the Globus toolkit could contribute in our framework to the implementation of membership, resource management, and monitoring services. Overall, the current Globus toolkit falls short of providing an adequate basis for the construction of autonomic VO-management. First, its reliance of Web Services (in the spirit of Web Services Distributed Management (WSDM) [9, 10, 55]) does not provide the comprehensive view and configurability provided by our architecture-based approach and its reliance on a reflective

[1]The different “tiers” of feedback control loops can be characterized as follows. The *Monitor* tier collects data that characterize the behaviour of the managed system. The *Analyze* tier interprets the data, using knowledge of the managed system (e.g. for diagnosis purposes). The *Plan* tier determines a course of action, based on the results of the analysis. The *Execute* tier implements the plan, by sending commands to actuators of the managed systems. We sometimes refer to the Analyze and Plan tiers collectively as the *Decision* tier.

component-based model. Second, the basic capabilities provided by the Globus toolkit must at a minimum be complemented with a proper framework for the construction of VO-wide management functions and their automation, addressing in particular the broad questions identified in the introduction of this document.

The recent survey on data grids [53] provides a useful taxonomy to characterize both the architecture and the capabilities provided by various data grid infrastructures. Interestingly, the survey points out that none of the studied data grid infrastructures, including those such as that of the EGEE project [15] that rely on the Globus toolkit and the OGSA reference architecture, qualifies as supporting autonomic management. They all are classified as being *managed*, i.e. as requiring a lot of human intervention for a variety of VO-wide management activities, including resource monitoring, user authorization, and data replication. Several recent works have targeted the development of VO management infrastructure and services. A number of them target principally security issues and/or the creation of virtual organizations construed mostly as access control and authorization domains. Such works include e.g. ICENI from the EGEE project [15], the extensions to the PERMIS [11] authorization software developed by the DyVOSE project [48], the VOMS membership service from the European DataGrid project [2, 32], the organization-based access control for virtual organizations presented in [30, 31]. Other work emphasize different capabilities to support virtual organizations, such as e.g. policy management [54], service-level resource management [23, 39], membership and authorization management [27, 56], service deployment [20, 38], configuration management [12], trust management [43, 57]. Overall frameworks for supporting and managing virtual organizations are less common: one can mention the generic DVO framework outlined in [42], the reference architecture developed by the TrustCom project [43], the VGrid and MiG frameworks [25], the virtual organization framework built on top of the Vinca service composition language [45], the VRM framework (which targets principally the support of business transactions and business-oriented virtual organizations) [40], and the agent-based framework of the CONOISE-G project [35]. Also relevant, though targeting principally secure collaboration and the sharing of application objects is the work on Secure Virtual Enclaves [46]. These different works, however, to the exception of [42] (but which reports a design only, and does not explain how automated management functions are to be supported), focus mostly on issues dealing with membership, support for virtual organization contracts, authorization and access control management. Support for an automation of VO-wide management functions (such as e.g. deployment and configuration management, failure management) is not discussed.

This is in contrast with work on autonomic systems, which strive, by definition, to build such support. There is a growing literature on autonomic systems (see e.g. [29] for a survey), but we can single out four different works which are closer to the framework proposed in this document, since they also adopt a similar control-based and component-based approach to autonomic systems construction. These are: the AutoMate system [1, 33] developed at Rutgers University and the University of Arizona, the KX system [24, 34] developed at Columbia University, the Rainbow system [21] developed at Carnegie Mellon University, and the work on architecture-based and planning-based deployment and automatic fault recovery at the University of Colorado [4, 3]. The latter work exploits software architecture descriptions and an AI planner for automating deployment and fault recovery. The control architecture is similar to that presented in this document and the other works above, and the component model this work relies on can be seen as a subset of the Fractal component model adopted in this document (see below). This work targets mostly single domain and cluster-size systems, and does not consider the range of services required for the autonomous operation of a virtual organization. One could probably adapt this planning approach to extend the decision element in our framework, but it is not clear how scalable it is. The KX system has been developed as a generic framework for “autonomizing legacy systems”, i.e. adding autonomic capabilities to target legacy systems. The KX architecture comprises: sensors and effectors that use a set of generic primitives for observation and actuation [52], an event bus that transports notifications from sensors to the control part of the architecture, a control part that comprises an event distiller for performing temporal event pattern analysis and correlation between multiple event streams, and a workflow engine, called Workflakes [51], that constitutes a decentralized process enactment engine. The overall KX architecture is similar to that presented in this document, but differs on several points, notably: its underlying component model and supporting actuation primitives are more limited than the Fractal model used in this document and its reflective capabilities, and it does not encompass management services required for the operation of virtual organizations. The Rainbow architecture is organized in three main layers: a system layer provides basic sensors and effectors on the managed system; an architecture layer that implements the decision tier of a feedback control loop, using a software architecture model of the managed system that provides a global perspective on the managed

system; and a translation layer that translates between system layer events and architectural layer events. In contrast to Rainbow, we advocate in our framework a component-based system construction or a component-wrapping approach to the handling of legacy systems. This makes the development of an autonomic structure more uniform, since the same component model can be used both for the construction or abstraction of the managed system, and for the construction of autonomic managers (implementing the decision tier of management control loops), and allows a uniform handling of deployment and configuration management. The AutoMate framework comprises four different layers: the system layer extends core OGSA services (security, resource and data management) to support autonomic behaviour, and contains specialized services including a peer-to-peer messaging service; the component layer supports the execution of autonomic components, providing services such as discovery, factory, lifecycle, context, which also build on OGSA services; the application layer builds to the component and system layer to support component composition and interaction; the agent layer comprises decentralized peer agents that support autonomic behaviours, including context-awareness agents, deductive agents whose rule-based collective behaviour provide the decision making capability to enable autonomic behaviour, and trust and access control agents. The framework presented in this document shares several elements with AutoMate, however we rely on a domain structure to provide scope for VO-wide management policies, and we make a more systematic use of (hierarchical) component-based construction, which leads to a more uniform and more scalable design. In particular, our design makes effective the construction of self-healing systems, as reported in [6].

The framework described in this document is an outgrowth of the Jade framework developed by the authors [6]. Compared to the original Jade framework, the current document considers potentially multi-domain structure for virtual organizations, considers management services (membership and security) that caters to the need of virtual organizations (which the original single domain Jade framework did not consider), and refines the design of the so-called autonomic manager component with the addition of rule-based decision making and workflow coordination for decision enactment.

2.3 Approach

2.3.1 Virtual organizations as autonomic systems

A virtual organisation integrates services and resources across distributed, heterogeneous, dynamic organisations to allow service and resource sharing when cooperating on the realisation of a joint goal. Each of these organisations is a management domain under the control of another management domain which represents the virtual organisation. Thus a virtual organisation is primarily a management domain that controls and coordinates the services and resources provided by others management domains to achieve a common goal.

The goal of *autonomic computing* is to automate, at least in part, the functions related to systems administration. This effort is motivated by the increasing size and complexity of the systems and applications, which has two consequences: (i) the costs related to administration are taking a major part of the total information processing budgets; and (ii) the difficulty of the administration tasks tends to approach the limits of the administrators' skills.

Autonomic computing aims to provide systems and applications with self-management capabilities. Self-configuration is achieved through automatic configuration according to specified policies, self-optimization through continuous performance monitoring, self-healing through detection of defects and failures, and taking corrective actions, and self-protection by taking preventive measures and defending against malicious attacks. Currently, human administrators perform management actions to ensure the desired operation of the system, using appropriate tools. One approach to autonomic computing, which we can call the *control approach* to autonomic computing, views the functioning of an autonomic computing system as an evolution of this practice, along the following lines:

- The overall management goals are expressed at a high level, and their translation into technical terms is performed by the management system.

- The management system observes and monitors the managed system; the observation may be active (triggered by the observer) or passive (triggered by the observed element).
- On the base of the observation results, the management system takes appropriate steps to ensure that the preset goals are met. This may entail both preventive and defensive actions, and may necessitate some degree of planning.

In a more detailed view, an autonomic computing system is organized according to the above overall scheme, sketched on Figure 1.

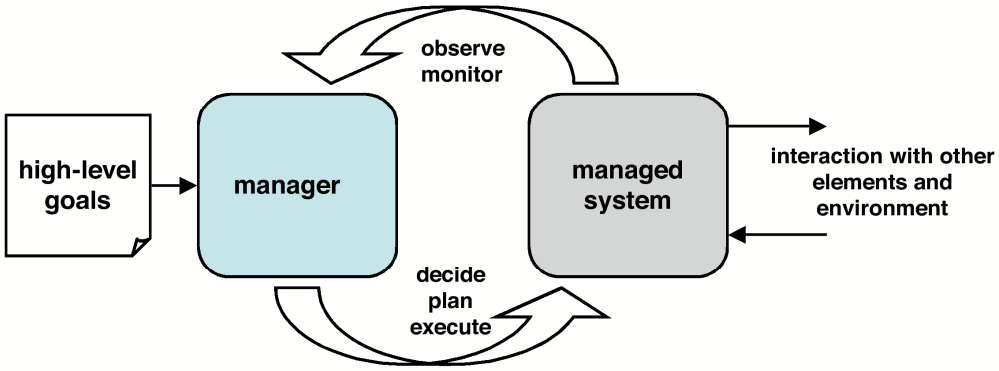


Figure 1 Overall view of an autonomic element

In this approach, autonomic computing is closely related to control. More precisely, an autonomic system is governed by a feedback control loop (feedback is preferred to feed-forward because it takes into account disturbances in the expected behaviour of the controlled system). In the following we describe the design principles used to provide a virtual organization management framework.

2.3.2 Architecture-Based Management

The notion of a system model is taking an increasing importance in the design, the development and the management of software systems. A *system model* is a formal or semi-formal description of a system's organization and operation, which serves as a base for understanding the system and predicting its behaviour, as well as for its design and implementation. The goal of current models of systems architecture is to describe a complex system as an assembly of elementary parts, using the notions of components, connectors, and configurations. These entities have different concrete representations according to the specific architectural model being used, but share common properties.

As system architecture is pervading the area of systems design, it has been realized that its constructs also form an adequate base for systems management. In particular, components may be conveniently used as units of deployment, of fault diagnosis, of fault isolation, as well as domains of trust; reconfiguration is adequately represented by component replacement and connector rebinding. The notion of *architecture-based management* captures this trend. It promotes the use of architectural models and formal or semi-formal system descriptions as guidelines for various management functions. Such descriptions are becoming commonly available, e.g. in the form of an Architecture Description Language (ADL), a framework for a formal description of a system conforming to an architectural system model.

2.3.3 Fractal-based Autonomic Computing

Our approach to an architecture-based, control approach to the design and construction of autonomic systems relies first on the choice of an appropriate *component model*. The component model we use is the Fractal model. The Fractal component model [7] is a reflective component model intended for the construction of dynamically configurable and monitorable systems. Its main features include: composite components (to obtain a uniform view of applications at various levels of abstractions), binding components

(to reify arbitrary connections and communication semantics between components), and introspection and reconfiguration capabilities (to monitor, control and modify the execution of a running system).

The choice of Fractal is motivated by several considerations:

- Fractal supports a hierarchical modelling of systems, allowing descriptions of a system at different levels of abstractions.
- Fractal supports software architecture descriptions with component sharing (where components may belong to different component hierarchies), which allows more natural specifications of system structures, and a separation of concerns in architecture descriptions.
- Fractal supports reflective components, i.e. components equipped with a meta-level structure that allows monitoring and controlling the execution of a component. The interfaces that make up a reflective component meta-object protocol in effect provide management interfaces for the component.
- Fractal is an open model, which makes no predefined choice concerning the semantics of component composition. This allows a system designer to define the semantics of a component binding or of a component meta-level best suited to its design.
- Fractal comes equipped with an extensible architecture description language (ADL) that can be used as a pivot language for capturing information related to different management concerns, and for expressing management actions.

We briefly present in the rest of this section the main features of the Fractal component model. More details on the Fractal model can be found in [7, 8].

A Fractal component is a run-time entity that is encapsulated and communicates with its environment through well-defined access points called interfaces. The Fractal component model defines two kinds of components: primitive and composite. Composite components encapsulate a group of other components which allows dealing with them as a unique entity.

Fractal components communicate through explicit bindings. A binding corresponds to a communication path between two or more components. Bindings are reified as first class components in Fractal, which allows for the construction of bindings as composite components, and with different communication semantics.

Composite components in Fractal have a reflective structure that is organized into membrane and content. The content of a composite component corresponds to the base level of the composite, and comprises all its subcomponents. The membrane of a composite corresponds to the meta-level of the composite. Elements of the membrane are called controllers. A controller is a meta-level object in charge of some aspect of control of the composite execution. Importantly, the structure of a membrane in Fractal is not fixed. New controllers can be defined by a Fractal designer or programmer to fit the needs of an application or system.

The Fractal specification defines several useful controllers: the content controller allows listing, adding, and removing subcomponents in the parent component; the binding controller allows binding a component interfaces to enable communication through binding components; the life-cycle controller allows starting and stopping the component; the Attribute controller allows setting and getting configuration attributes.

2.4 Virtual Organization Management Framework

This section describes a framework which can be reused and specialized for specific use cases. The goal of this framework is to ease the construction of a virtual organization as an autonomic system while reusing a set of common management services. The framework comprises several elements: (1) a *domain structure*, which provides scope for VO policies, and allows for the construction of virtual organizations as hierarchies and federations of virtual organizations; (2) a component-based notion of *resource*, which subsumes notions of information processing resources and services in other virtual organization and grid frameworks; (3) a set of core *management services* which provide basic infrastructure functions for the construction of virtual organizations; (4) a notion of *autonomic manager* for the construction of automated management functions within virtual organizations. We consider these different elements in turn.

2.4.1 Domain structure

The notion of virtual organization is primarily associated with the management of relationships between users, organizations and resources. It is then only natural to associate a virtual organization to a management domain. The notion of *management domain* has been introduced in [49] to provide an explicit scope for the definition of management policies and management operations. In particular, a management domain constitutes a naming context for the entities that belong to the domain and that can be the objects or the principals of management operations. As our first design decision, therefore, we associate a VO with a single management domain that encompasses as its entities all the (physical or software) resources pooled by the VO, the users and the organizations participating in the VO, the latter being themselves modelled, recursively, as VOs. Following our component-based approach, all the entities in a VO and its associated management domain, are modelled or implemented as components.

Figure 2 illustrates that a VO comprises components of three different kinds: managed resources, autonomic managers and management services. The extent of a VO management domain is given by a set of physical nodes (i.e. computers running VO components), called *managed nodes*. Note that, because of self-management in a VO, managed nodes can support the execution of both managed components, and management components (i.e. components involved in the implementation of management functions). Managed nodes themselves are examples of managed resources in a VO management domain. In the rest of this report, we use the term node for managed node, unless explicitly specified (e.g. physical node). To enable self-management functions, autonomic managers and components that provide core management services can themselves be managed resources in our VO architecture. To allow a VO management domain to scale with the number of managed nodes, the managed nodes in a VO management domain are organized as an overlay network.

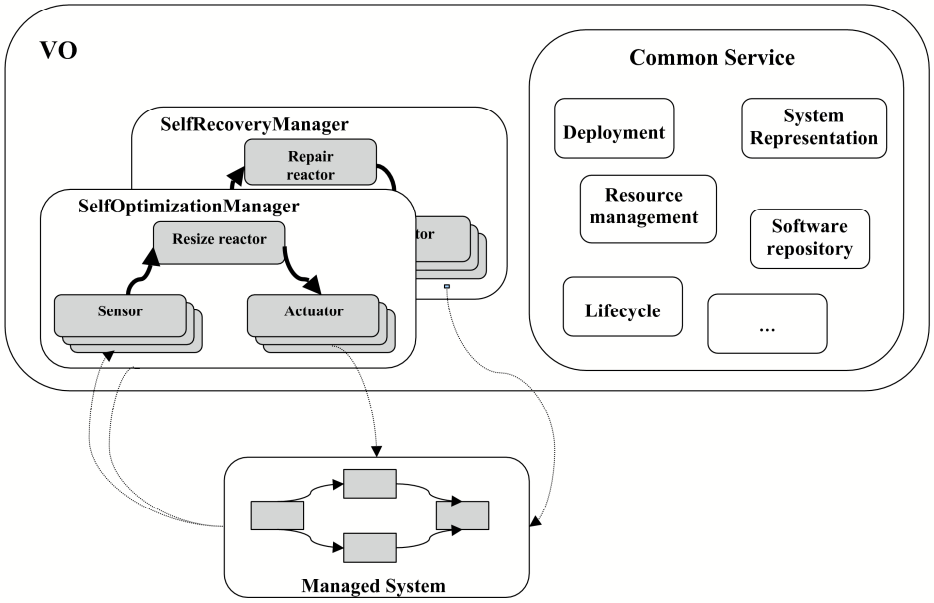


Figure 2 A virtual organisation management system

One can envision three different kinds of relationships between VOs and their associated management domain:

1. a *delegation* relationship [22], allowing in particular the construction of hierarchically organized VOs, with subordinate VOs being delegated responsibility for managing a subset of the parent VO resources.

2. a *peering* relationship, allowing in particular the construction of a VO as a federation of cooperating VOs.
3. a *participant* relationship, where a VO is a contributing resources to another VO.

2.4.2 Resources

A Resource is any piece of a system required to provide a service. It can be a hardware element such as a cluster node, or software element such as a middleware running on a node or an application running on the middleware.

Our framework is based on an architecture description language to describe resources. From this description, the management system will update a runtime system representation which represents the current system infrastructure in terms of resources and their relationships. The system will also create a runtime entity which represents the managed resource and which provides a management interface to control the resource.

Resource description

In order to be manageable by humans or by VOs, a resource has to provide some information about itself. This information comes from a resource description which describes meta information about the resource. Such descriptions include the following data:

- Resource attributes represent any kinds of property regarding the local execution of the resource such as configuration properties and environment variables.
- Resource relationships represent dependencies between resources. These dependencies are represented as a binding between the resources. For instance a web server is a resource that depends on another resource which represents a database.
- Resource containment represents the resources embedded in another resource. For instance a virtual machine which embeds a tomcat server is represented by a resource (the virtual machine) which contains another resource (the tomcat).
- Resource interfaces represent the functional interface of the resource (if the resource had one) and its management interface.
- Resource names identify resources.

A resource can provide additional information specific to the resource if necessary. Resources are described using an architecture description language. We choose to describe resources using the Fractal ADL because it provides the required construction for resources description: (i) type definition, (ii) type inheritance, (iii) properties configuration and introspection, (iv) containment relation, (v) binding relation, (vi) control operations definitions. Furthermore from such description, the ADL factory can deploy the runtime entity that will implement resource description and control. This means that resources are remotely managed at runtime by a component that represents the resource. Implementing resource description and management as component is a key to the decentralize resources management and control.

From a language point of view, the ADL language is not a fixed language, but a set of ADL modules from which various ADLs can be constructed. The idea is to do aspect oriented "programming" at the ADL level: each module is intended to correspond to an ADL "aspect". Since the Fractal ADL does not impose a concrete syntax, the common representation used for interoperability between Fractal ADL tools is abstract syntax trees (AST). An AST provides both a generic API, similar to DOM (but not DOM itself, in order to be independent of XML), and a typed API. In our case, specialization of the Fractal ADL will be done to allow resources description. Each module represents a part of a resource description.

Notice that resources can have various level of granularity (e.g., a processor, a virtual machine, a physical machine, a cluster, a router, a network, a web server, etc.). Whereas resources can be classified in hardware resources, middle-ware resources and application resources, the language used to describe resource is unified thank to Fractal ADL. Furthermore from an ADL description a runtime element can be generated to hide resources heterogeneity from the management point of view. Figure 3 gives an example of

three complex resources: (i) a network composed by an internet gateway, gigabit switch and nodes, (ii) a cluster composed by nodes, (iii) a J2EE infrastructure composed by web server, EJB server and database.

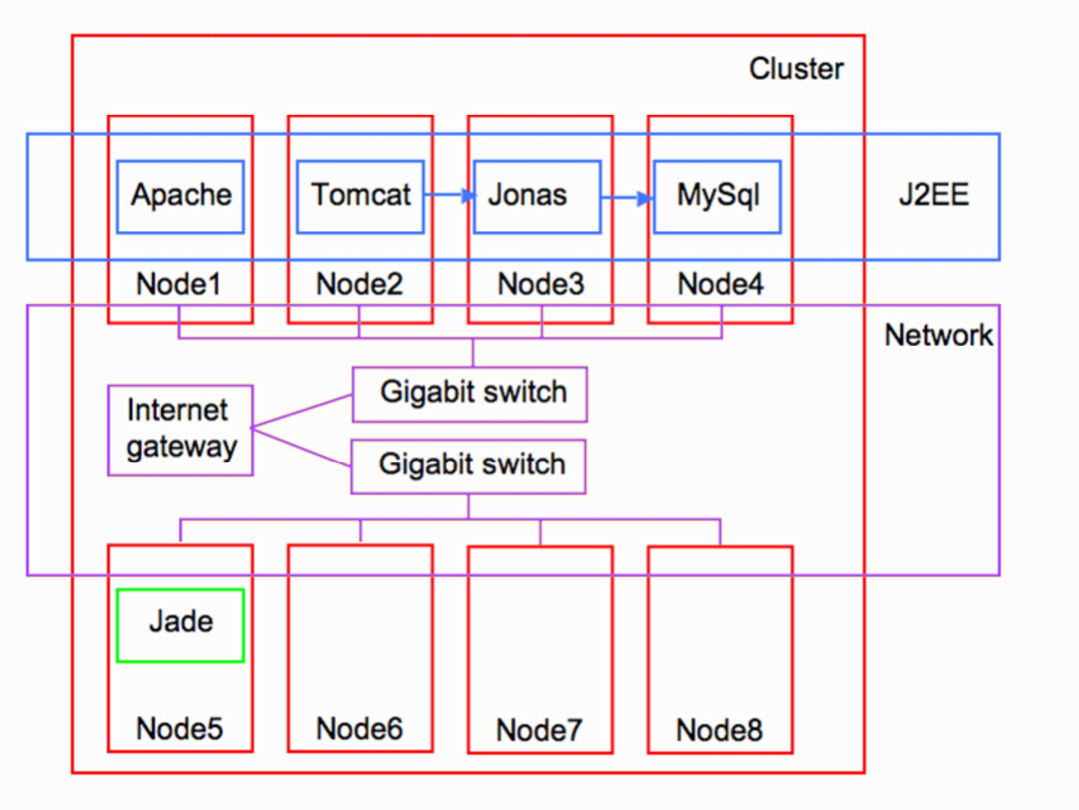


Figure 3 A complex system description

In the following we give some examples of resources descriptions. The description of a resource requires two steps:

- The description of the resource's type. This description can specify the management interfaces provided by the resource as well as its functional interfaces (provided and required) by the resource (the interface tag). In the current Fractal ADL, interface's signature is given by using java interfaces. However, this is not mandatory. The signature of services can be of any kind. In the case of Grid4All, we can use the WSDL language to describe such interfaces. Furthermore a type description can also contain the class providing the implementation of the management operations provided by the resource (content tag). It may also contain the definition of attributes which will be inherited by all the instance of the resource.
- The description of an "instance" of resource. This description specifies the name of the resource and some runtime properties related this particular instance. In the following we describe an instance of web server resource called "apache1". In this example we set the local working directory (/tmp/j2ee) for the web server and its port (8081). Furthermore we describe via the package tag the binary unit required to install the webserver (this information will be used by the deployment service to deploy this resource if necessary).

In the following, we give the description of a type for node.

```
<resourceType name="node" controller="ParametricCompositeResource">
        <resource name="ram1">
                <resourceType name="ram"/>
        </resource>
        <resource name="disk1">
                <resourceType name="disk"/>
```

```
        </resource>
          <resource name="proc1">
                <resourceType name="processor"/>
          </resource>
          <resource name="OS1">
                <resourceType name="system"/>
          </resource>
    </resourceType>
```

The following gives an example of resource instances:

```
    <resource name="node1"  definition="node">
          <resource name="OS1" definition="system">
                <type name="Windows" />
                <resource name="filesystem1"
                     definition="FileSystem">
                      <attributes>
                        <attribute name="type" value="FAT"/>
                              <attribute name="dir" value=”/school/”/>
                        <attribute name="amount" value="2000"/>
                      </attributes>
                </resource>
           </resource >
           <resource name="proc1" definition="processor">
                 <type name="intel"/>
           </resource >
          <resource name="ram1"  definition="ram">
                <attributes>
                      <attribute name="type" value="ddr"/>
                      <attribute name="amount" value="2000"/>
                </attributes>
          </resource>
          <resource name="disk1" definition="disk">
                <attributes>
                      <attribute name="type" value="scsi"/>
                      <attribute name="amount" value="2000"/>
                </attributes>
          </resource>
     </resource>
```

Resource Wrapping

Wrapping is one of the common services of the Grid4All architecture. Wrappers can be used to encapsulate diverse software resources so that they all present a common and simplified interface. Software wrapping is

a technique in which an interface is created around an existing piece of software, providing a new view of the software to external systems, objects, or users. Using wrapping techniques, legacy software (e.g. Apache, Tomcat, MySQL etc) can be changed to software components that can be integrated (to new computing environment), manipulated (installed, configured, re-configured), controlled and maintained (repaired) automatically and in efficient manner. In the context of Grid4All, wrapping helps developing an environment whereby a potentially large number of existing legacy software components can be adapted for use within a new software administration framework. Wrapping facilitates self-managing capabilities such as self-configuration, self-optimization, self-healing, and self-protection which are very important properties of autonomic computing.

Basic principles of wrapping

The aim of wrapping, in the context of Grid4All, is to encapsulate the different legacy software platforms in order to develop administration systems with capabilities of performing the resources management activities in automatic, efficient and dynamic fashion. The basic idea of wrapping is to provide control mechanism to do the above activities by the system. The internal activities of the legacy systems or the interactions/calls between them are not intercepted or controlled by the system. The system facilitates only the configuration of the different components.

Fractal model and wrappers

Since the Grid4All system is built on the Fractal component model, the purpose of the wrappers is to create a corresponding Fractal component for each component of the legacy software. The Fractal model provides adequate support to develop wrappers for legacy software components because it provides well-defined interfaces such as attribute controller, binding controller, content controller and lifecycle controller which can easily be mapped to the installation, deployment, configuration and maintenance activities in the legacy software. Once one legacy software component is wrapped into a Fractal component, the manipulation of the legacy software is done through the different control interfaces of the Fractal component.

Consider a clustered J2EE architecture composed of Apache, Tomcat and MySQL components. In such a scenario, normally, a human administrator configures the architecture by accessing the configuration files associated with each component. For example, to configure the port of the Apache server and to define the binding or connection of the Apache server with other components, this requires the modification by the administrator of the httpd.conf and the worker.properties files respectively. With the Grid4All infrastructure, by wrapping each of the legacy software into Fractal components, those activities can be done in an automatic and dynamic manner by an administration system.

For example by creating a Fractal wrapper for the Apache server, the attribute controller interface is used to configure the properties of the Apache server and the binding controller interface is used to define the connection of the Apache server with other components of the system, in this case Tomcat servers. At the same time the life cycle controller interface is used to start, stop and read the state (running or stopped) of the server.

Related work on wrappers

We identify the following related works on wrapper development, which is a potential source of inspiration.

- *Kilim* (http://kilim.objectweb.org/). Kilim is a configuration framework based on the Fractal component model that provides a generic model, a powerful language and tools (a parser, a runtime configuration viewer) to facilitate, automate and control the configuration process of arbitrary complex applications. Kilim enables definition of composable abstractions called templates, capturing encapsulated sub-systems, defining their properties and their connectivity (slots), defining the mapping of these abstractions to existing code in terms of constructors used to create the various instances they may contain and setters/methods used to configure and to connect them; and the recursive assembly of these components allowing to build complex systems. Here, to build a complex system a template is created by assembling existing templates. Kilim creates components from Java code using the templates.

- *Cargo* (http://cargo.codehaus.org/Home). Cargo is a thin wrapper around existing containers (e.g. J2EE containers). It provides different Java APIs to easily manipulate containers: starting/stopping containers, configuring containers for deployment on any user-specified directory, deploying WAR and EAR components on these containers.

Requirements

We aim at providing a wrapping service with two components: a wrapper description language (WDL) for the specification of the wrapper associated with a legacy component, and a wrapper generator (WG) which generates the wrapping software from the WDL description. The specification provides vital information that is required to generate the wrapper of the legacy software. The wrapper generator creates the appropriate Fractal component that corresponds to the legacy software based on the specification/description.

To aid the development of an appropriate scheme for the wrapping activity, the following requirements should taken be into account.

- **Dedicated language:** To facilitate the definition of wrappers, a dedicated language is required to specify the translation of administration interfaces into legacy software administration functions (which are proprietary). The administration interfaces of the wrappers are used by the system to control and configure the legacy software.
- **Extensibility:** Since the wrapped legacy software is very heterogeneous (very different administration functions), the WDL should be extensible in order to accommodate very different classes of legacy software (and classes of administration functions). However, we believe that few personalities of WDL should cover most of the application domains.
- **Generate Fractal components:** The Grid4All platform is based on the Fractal component model. From a WDL specification, we must automatically generate the Fractal component that corresponds to the wrapped legacy software.

If the wrapper specification languages are generic, it has a wide applicability to different application domains but requires more effort from developers to program wrappers. More specialized languages require less programming effort but have limited applicability. The good trade-off is probably to provide a wrapping framework for defining specialized WDL languages (and their associated generators) for different applications domains.

2.4.3 Management Services Overview

This section gives an overview of the management services provided by the Virtual Organization. We depict the management services from a functional point of view.

Membership

The aim of the membership service is to manage users' registration and login. It keeps track of who is member of a VO, and of the current status of the users. In the context of P2P systems, a membership management algorithm, which provides each node with run-time peer sampling service, is essential for many peer-to-peer (p2p) network applications, such as gossip based broadcast algorithms, distributed hash tables, dynamic load balancing, random sampling, and network topology construction. Full membership management maintains the complete list of all network members at each node.

Membership in the Grid4All context

In the context of virtual organisations, the membership service (i) keeps track of who is member of a VO, and (ii) allows looking up users and their presence. The membership service provides operations for two kinds of

users: standard members and VO administrators. It allows administrators to grant/revoke permission for users to become members. It allows users to register, to log in, and log out. Users select their own usernames, which should be valid email addresses in order to be unique. When a user wants to become a member, the VO checks the user's authorization. By becoming a member, the user accepts its role and the policy related to it. When registering, the user provides a profile with information on itself and on the resources provided when the user logs in. The user also sets up a password maintained by the VO for the login step. This supposes that a user has obtained 1) the software package: Grid4AllBoot and 2) a ticket (e.g. web page e-mail, etc). The software is installed and provides a local proxy object that is used for further communication.

Requirements

The storage and communication requirements of full membership management algorithms grow linearly with the network size, which is prohibitive for large-scale applications. Furthermore, the membership service must check users' credentials and roles. Thus, the main requirements for this service are decentralisation and security.

Life Cycle

The aim of this management service is to control the lifecycle of the VO and the lifecycle of the resources belonging to the VO. This service is very important since it will coordinate the creation of a VO and the availability of all the VO's services. In the following paragraph we describe the basic states related to lifecycle. These states are both valid for the VO software and resources:

- Deployed: Means that the VO is deployed but the services provided by the VO are not available yet.
- Manageable: Means that the VO is deployed and the management services are available.
- Started: Means that the VO is fully operational and all the services are available.
- Stopped: Means that the all services except the management services are stopped. A stopped service is unavailable.
- Updated: Means that the VO software has been successfully updated.

Requirements

This lifecycle has to be provided by resources at various level of granularity. For instance, we can remotely stop (i) a machine which implies stopping all the software running on the machine, or (ii) a whole cluster which implies stopping all the machines belonging to the cluster. This is a stringent requirement because implementing lifecycle operations on a complex resource (such as a cluster) requires implementing distributed workflow, potentially on a large-scale environment.

Resource management

Resource Management is a central component of a VO. It involves managing resources in the system. Resources include traditional resources like compute cycles, network bandwidth, space or a storage system and also services like data transfer, simulation etc. Its basic responsibility is to provide resources discovery and resources allocation. Resources customers can be either the VO's members or the VO runtime itself. This service keeps track of the resources provided and used by members. The customers essentially interact with the resource manager that hides the complexities of grid computing. The resource manager (i) discovers resources that the customer can use, and (ii) allocates the resources that match some constraints, such as job execution deadline, and the maximum cost of execution. If the allocation constraints cannot be fulfilled because there are not enough resources, the resource manager has two basic policies:

- It can propagate the request to the resource market. Thus the basic policy of the allocation subsystem is to allocate resources belonging to VO's members before requesting from the market. All

interactions between the resource manager and the market go through a "buyer agent" embedded in the resource manager.

- It can allocate part of the resources that are currently available. It will notify the resource customer when additional resources that match the constraints become available. This policy uses a call-back API that has to be implemented by resource customers.

Requirements

Resource Management in Grid systems is complex due to various factors discussed in the following. Grid resource management systems have to deal with different site authorities. Traditional resource management systems work under the assumption that they have complete control on the resource and thus can implement the mechanisms and policies needed for effective use of that resource. But in Grid systems resources are distributed across separate administrative domains. This results in a complex environment owned by different individuals/organizations, each having:

- Their own resource management policies and different access-and-cost models. Different local resource managements systems like NQE, LSF etc can be employed. A Grid resource management system should be able to interface and interoperate with these local resource managements.
- Their own machine, network and system. Thus the resources management have to interoperate with very heterogeneous environment and resources.
- Their own security mechanisms.

Another major issue to designing a resources management service is the dynamicity and the scale of the environment. Resource management has to deal with a huge number of resources with a high dynamicity in resource presence. Maintaining a full map of available resources is not obvious. The design of the resource management service should make a trade-off between consistency of resources management information and the efficiency of the resources discovery and allocation. To deal with the heterogeneity of the Grid4All environment, we propose to use a wrapping layer to have a homogeneous view of managed resources in terms of control operations that can be applied on the resource.

To summarize, the resource management service provides the following features:

- Discovery: It provides the functionalities to lookup resources according to some constraints. In a first step, these constraints are expressed as a set of attributes that must match resource descriptions. We have currently two kinds of constraints: constraints on the resource type (for instance a node with 5GO of RAM), and on the period of time the resource needs to be allocated (the lease).
- Allocation: It provides the remote ability to allocate a resource for the need of the VO. The resource manager can be configured (i) to request resources to the market if there is not enough resource in the VO (ii) to fulfil an allocation request in multiple steps by notifying resources customer when more resources become available.
- Resource addition/deletion: it provides the ability for a resource owner to add new resource to the VO and to remove a resource from a VO. A resource owner is typically a member of the VO.

Deployment

The goal of the deployment service is to install and configure applications in the VO as well as to undeploy them. The deployment service will deploy application’s components according to (i) the resources provided by members and (ii) some constraints related to the application (for instance, locality constraints). In this section we provide a functional description of the deployment service.

The deployment service is architecture-based. This means that given a description of the software architecture, the deployment service is capable of installing, instantiating and executing software components described by this architecture. Moreover, the deployment system allows for component versioning and dynamic updates; several versions of software components can coexist, and components can also be replaced with newer versions.

The software elements to be deployed must be described using an ADL as shown in section 4. The next section presents the requirements of the deployment service. Using a standard ADL factory does not currently fulfil all these requirements.

Requirements

- The user can insert or remove a component into an existing component structure.
- The configuration of a component can be explicit or implicit.
- The deployment service must be able to deploy complex component structures which can be made of legacy components.
- Non functional properties can be easily plugged in the deployment process. For instance an interesting property is to ensure the atomicity of deployment orders.
- The deployment service must be fully specializable. In particular, the implementation and the scheduling of the deployment orders can be controlled by the user.

Deployment API

The API of this service is the following:

Object newComponent (final String name) // name is the filename of the ADL file

Deployment workflow

This service is implemented using an ADL Factory. The deployment algorithm requires that physical nodes are wrapped as managed elements and provide (i) an installation API to install software on the node and (ii) a component factory API to create managed elements. The algorithm of the newComponent method is summarized below:

- Lookup the node where the component must be deployed.
- Install the component package on the target node
- Instantiate the component on the target node
- Configure the component (i.e., set its attributes and its external bindings). This step uses the managed element API.
- Process recursively the sub-components, if necessary.

The basic deployment steps are depicted on Figure 4. The design of the deployment system will be detailed in a following section.

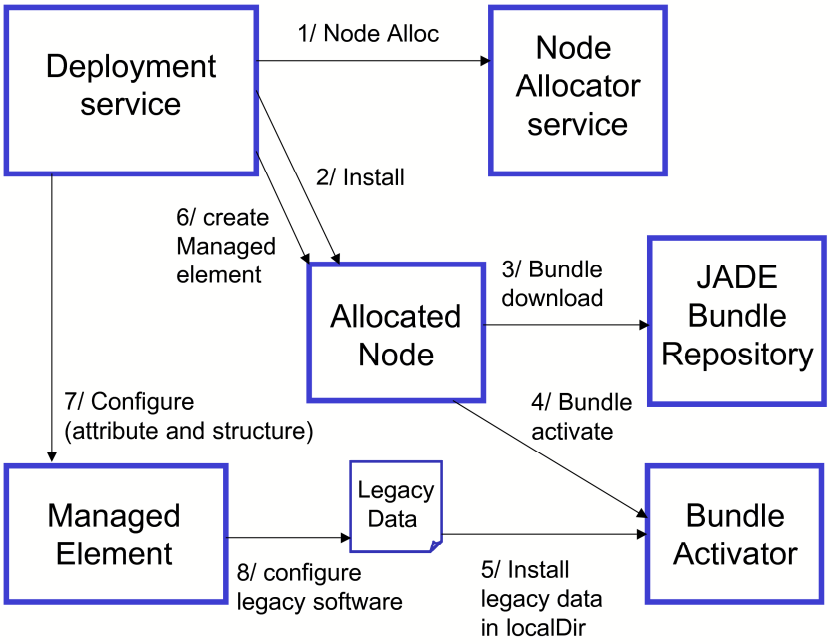


Figure 4 The steps of the deployment process

Monitoring

This service aims at defining the monitoring requirements for the Grid4All platform in terms of system observation and information feedback. These requirements come from the project's autonomic scenarios in the fields of self-repair, self-configuration and self-optimization. Roughly, the Grid4All autonomic infrastructure needs to know what the system it is managing is made of, and what is the state of this system. This information quest may follow a variety of modalities according to the target resources (software elements, hardware elements, network, execution support abstractions, etc.) and the target features (deployment, configuration, repair, or optimization). One of the key ideas behind this requirement is that it should also benefit from the autonomic management features of the Grid4All platform.

Required features

The monitoring feature aims at observing the managed system and delivering the observation results to the autonomic control features, either by broadcasting events (push mode), or by keeping the information that will be picked by the interested elements when needed (pull mode). For instance, an autonomic control element may be interested in receiving an event when a system load threshold is reached, while periodically checking the availability of a computer on the network.

Elements of the monitoring functionality, that we will call probes, are likely to be active, possibly to get information (observation, measurement), and possibly to send the extracted information. Probes may also require a memory feature to keep the extracted information during a certain amount of time, either as raw data or as statistical values on sliding time frames.

Scenarios about self-repair basically need fault detectors, such as heart beat (I'm alive!) or ping (are you alive?). Self-optimization scenarios need:

- System load measurements (CPU consumption, RAM utilization, disk and network transfer rates, etc.),
- Observation of middleware and applications involved in scenarios.

Aggregated observations

Beyond those basic observations that are obtained from basic probes dedicated to specific software or hardware elements of the managed system, there is also a need for more elaborate, higher level indicators computed from the basic observations. For instance:

- A system load or an application server load metric
- An aggregation of CPU loads for a given cluster of computers

Both kinds of observation may be required at the same time, and may evolve in time. Moreover, it might be necessary, or at least convenient, to share probes among several combinations. For instance, a CPU probe may be used as is, for local CPU observation, as well as simultaneously in a local system load aggregation and a CPU load aggregation for a cluster of computers. Finally, probes and aggregations should be reconfigurable.

Uniform probe representation

The architecture should feature a uniform representation of all probes. This means that all probes must expose identical interfaces, even when they were in charge of monitoring different system properties and/or heterogeneous resources. The monitoring framework should support various probe types for monitoring different system resources and properties, at different abstraction levels. For example, probes could be available for monitoring a system's CPU, a JVM's memory, an application's workload, or a cluster's general load. However, all probes should be equally accessible via identical interfaces, in order to retrieve monitoring data or apply control commands.

More generally, the monitoring functionality must be deployable and manageable similarly to any other element in the Grid4All platform. As a consequence, we must rely on a uniform architecture based on Fractal components.

Recursive, hierarchical probe composition

The architecture must support probe organisation into recursive, hierarchical constructs. This allows system managers to build arbitrary monitoring hierarchies based on individual probes. Probes can be basic or composite. Basic probes extract actual data from the monitored system elements and represent leaf nodes in the monitoring hierarchy. Composite probes collect data from lower-level probes, which can be in turn basic or composite. Collected data is being processed so as to provide a higher-level monitoring view of the corresponding resources.

Scalability

The monitoring framework should scale gracefully with the number of monitored resources and aggregated data sources. This means that the overall monitoring hierarchy should withstand increasing numbers of monitoring nodes and that each composite probe should be able to handle large numbers of data sources.

Performance

The performance overheads induced by monitoring probes on the managed systems should be minimized. A popular approach is to minimize overheads caused by instrumentation code (i.e. for basic probes) on the actual monitored nodes and to use separate stations for performing the remaining data processing and management functions. In addition, the performance of data transmission and processing procedures should be such that the Grid4All framework can effectively learn of relevant system changes and react in due time.

Component-based architecture

The monitoring service by itself is likely to become a complex, distributed infrastructure by itself that would deserve relying on the autonomic services of the platform. To achieve this, the monitoring service shall be based on components that conform to the Manageable component type. Moreover, functional requirements in terms of composition (for aggregate observations), sharing and runtime reconfiguration also advocate for an advanced component model featuring recursion, sharing and reflection.

2.4.4 Autonomic Manager

An autonomic element implements a control loop that regulates a part of the system, which we call the *managed system*. The managed system consists of a collection of *managed elements*. A managed element (ME) may in turn consist of a single elementary hardware or software component, or may be a complex system in itself, such as a cluster of machines, or a middleware system. In order to be included in a control loop, a managed element must provide management interfaces, which include sensor interfaces and actuator interfaces[2] . These are used by a controller, also called an *autonomic manager* (AM), to regulate the managed system through a feedback control loop. The *autonomic element* (AE) is the ensemble including the managed elements and the control loop, i.e. the controller and the communication system that it uses to access the management interfaces (Figure 5).

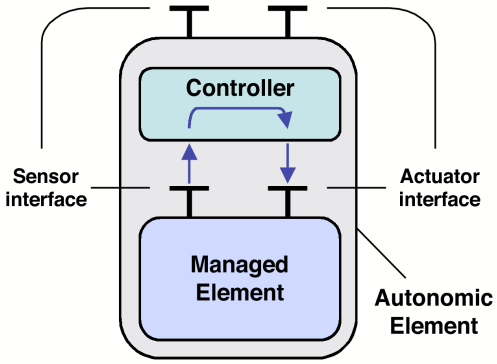


Figure 5 An autonomic element

An autonomic manager may itself be equipped with management interfaces, thus becoming in effect a managed element. This allows a hierarchical organization of AMs. In the same vein, an elementary managed element, such a hardware device like a disk unit, may itself include embedded, built-in control loops, making it an autonomic system, even if these control loops are not directly accessible through the ME's management interfaces.

The controller that regulates a ME in an autonomic element usually deals with a single control aspect, e.g. security, fault tolerance or performance. A given ME may then be part of several AEs, each of which deals with a specific aspect; each of these AEs has a specific AM and may be regarded as a different management domain (Figure 6).

[2]The complexity of these interfaces depends on that of the managed element.

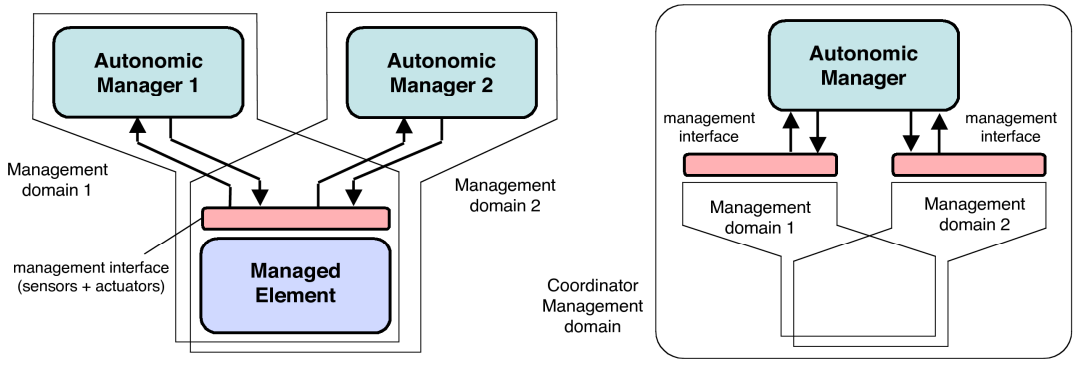


Figure 6 Multiple domains for autonomic management

The AMs managing different aspects of a common element have different criteria and may take conflicting decisions. An approach to resolving such conflicts is to coordinate the AMs that manage different aspects through a new AM (the coordinator), which applies a conflict management policy (Figure 6). An example of such a policy might be as follows, to arbitrate between a repair manager and a performance optimizing manager: give priority to the repair manager, except when this would degrade the quality of service of a specified client. Typically, the coordinator controls the decision making of managers, which then act directly on the managed system. Alternatively, the coordinator may intercept actuator commands initiated from managers and then decide whether/how to perform them on the managed system.

The notion of an autonomic element is thus seen to cover a wide range of situations, at different levels of granularity. In the above discussion, we came across three typical levels, in increasing order of abstraction:

- An elementary component, with embedded internal control.
- A mid-level manager, at the application or middleware level, controlling an aspect such as QoS, security, or fault tolerance.
- A coordination manager, whose role is to arbitrate between several aspect-specific managers in charge of a common set of resources (a shared ME).

Autonomic Manager Framework

An autonomic manager (AM) is a component that implements the analysis, planning and execution stages of a control loop: it monitors a set of managed elements, analyzes notifications coming from managed element sensors, diagnoses the state of the system, decides on or plans a course of action in response to the diagnosis and according to high-level administration policies, and executes the corresponding command plan. The managed elements controlled by an autonomic manager may be designed implicitly or explicitly.

We define an autonomic manager by specifying:

- A set of event-condition-action (ECA) rules. These rules are implemented as control loops embedded in autonomic managers.
- The collection of elements placed under the control of a manager. These elements correspond to Grid4All resources. Thus a managed element (ME) may consist of a single elementary hardware or software resource, or may be a complex system in itself, such as a cluster of machines, or a middleware system.

In order to be included in a control loop, a resource must provide management interfaces, which include sensor interfaces and actuator interfaces[3] . These are used by the autonomic manager to regulate the managed system. A manager may contain rules for emitting events and processing events to/from other managers, thus enabling a hierarchical organisation of managers.

[3]The complexity of these interfaces depends on that of the resource.

As a first step to designing the autonomic manager, we evaluate rule engines for interpreting the global management rules that will be expressed in a high-level language close a natural language. We also consider workflow engines since the manager will use such an engine for implementing the plan function. The following sections give an overview of rules engines and workflow engines.

Rules engines

The underlying idea of a rule engine is to externalize the business or application logic. A rule engine can be viewed as a sophisticated interpreter of if-then statements. The if-then statements are the rules. A rule is composed of two parts, a condition and an action: When the condition is met, the action is executed. The if portion contains conditions (such as amount ≥$100), and the then portion contains actions (such as offer discount 5%). The inputs to a rule engine are a collection of rules called a rule execution set and data objects. The outputs are determined by the inputs and may include the original input data objects with modifications, new data objects, and possible side effects (such as sending email to the customer). Rule engines should be used for applications with highly dynamic business logic and for applications that allow end users to author business rules. A rule engine is a great tool for efficient decision-making because it can make decisions based on thousands of facts quickly, reliably, and repeatedly. Adopting a rule-based approach for an application has the following advantages:

- Rules that represent policies are easily communicated and understood.
- Rules retain a higher level of independence than conventional programming languages.
- Rules separate knowledge from its implementation logic.
- Rules can be changed without changing source code; thus, there is no need to recompile the application's code.
- Speed and Scalability: The Rete algorithm [16], Leaps algorithm [5], and its descendents such as ReteOO provide very efficient ways of matching rule patterns to domain object data. These battle-proven algorithms are especially efficient when datasets do not change entirely (as the rule engine can remember past matches).

Related work

There exist several Rule Engine implementations on the market. The following ones were considered for Grid4All because of their popularity, and varied target audience:

- JBoss Rules [16], which is free, open source, and distributable,
- Jess [36], which is free for personal usage and not distributable,
- Mandarax [37], which is open source.

They all implement the JSR-94 specification, which allows avoiding vendor lock-in. The specification does not encompass the expression language used to define the rules. Although this means that rules are expressed in different formats, the same concept is applied everywhere: conditions are expressed as properties of Java objects, and actions are expressed as Java code. There is currently some ongoing work to propose a common rule format. W3C is working on the Rule Interchange Format (RIF), and OMG is working on a standard format based on RuleML.

JBoss Rules is selected for the following advantages:

- Open source (required)
- Better performance compared to Mandarax
- Active and dynamic community of developers
- Eclipse plugin for editing the rules

JBoss Rules

JBoss Rules [16](formerly Drools) is a rule engine that uses the rule based approach to implement an expert system and is more correctly classified as a production rule system. The term "production rule" originates from formal grammar, where it is described as "an abstract structure that describes a formal language

precisely, i.e., a set of rules that mathematically delineates a (usually infinite) set of finite-length strings over a (usually finite) alphabet" (Wikipedia).

JBoss Rules has implementations for both the Rete and Leaps algorithms. The Rete implementation, called ReteOO, is an enhanced and optimised implementation of the algorithm for object-oriented systems. Other Rete-based engines also have marketing terms for their proprietary enhancements to Rete, like RetePlus and Rete III. It is important to understand that names like Rete III are purely marketing where, unlike the original published Rete algorithm [16], no details of implementation are published. The most common enhancements to Rete are covered in [14].

The engine is composed of three parts: the production memory, the inference engine, and the working memory. The production memory stores the production rules, and the inference engine matches the rules against facts. Facts are asserted into the working memory, where they may then be modified or retracted. A system with a large number of rules and facts may result in many rules being true for the same fact assertion; these rules are said to be in conflict. The inference engine manages the execution order of these conflicting rules using a conflict resolution strategy.

There are two methods of execution for a production rule systems: forward chaining and backward chaining; systems that implement both are called hybrid production rule systems. Forward chaining is 'data-driven' and thus reactionary - facts are asserted into the working memory which results in one or more rules being concurrently true and scheduled for execution - we start with a fact, it propagates, and we end in a conclusion. JBoss Rules is a forward chaining engine. A rule has the following rough structure:

```
rule "name"
        ATTRIBUTES
    when
        LHS
    then
        RHS
    end
```

LHS (Left Hand Side) is the conditional parts of the rule. RHS (Right Hand Side) is a block that allows Java semantic code to be executed

Workflow engines

A workflow engine, sometimes referred as a BPM (Business Process Management) engine, is a software component that breaks a work process down into tasks. A basic example of such a process is an approval workflow process, in which an employee needs a manager's permission before running an application. A workflow engine provides an infrastructure to model this workflow, execute it, assign the tasks to its participants, and monitor it. To achieve the desired results, it may interact with humans or machines through, for example, Web services. This enables integration with platforms different from Java, like mainframes or .NET.

Bonita workflow

Bonita is a workflow solution for handing long-running, user-oriented workflows providing out of the box workflow functionalities to handle business processes. Bonita is Open Source and is downloadable under the LGPL License (http://bonita.objectweb.org). Its main key benefits are:

- A comprehensive set of integrated graphical tools for performing the process conception and definition, the instantiation and control of this process, and the interaction with the users and other applications.

- 100% browser-based environment with Web Services integration that uses SOAP and XML Data binding technologies in order to encapsulate existing workflow business methods and publish them as JavaEE-based web services.
- A Third Generation Workflow engine based in the activity anticipation model. This flexibility allows a considerable increase of speed in the design and development phases of cooperative applications.
- Support of the XPDL standard, backed by the WfMC (Workflow Management Coalition).

The following section discusses the issues related to the use of the tools described just above in the context of Grid4All.

Issues

Designing autonomic managers raises the following major issues:

- The size of the system: The decision system has to enforce policy onto a large number of resources, members, and application service. Clearly a centralized decision system cannot fit the requirement of Grid4All context. A major issue is to scale the decision tools (such as JBoss Rules) onto this environment.
- The coordination of multiple policies: The autonomic manager that regulates managed elements usually deals with a single control aspect, e.g. security, fault tolerance or performance. The AMs managing different aspects of a common element have different criteria and may take conflicting decisions. This requires a way to coordinate decision and reaction in a wide environment. Since full coordination and consistency seems difficult to achieve in large-scale system, we have to identify the set of policies that require minimum coordination.

Policy Management in Grid4All

Autonomic managers are the basic tool (language and runtime) that can be used to implement the decision rules used to enforce VO's policies. A policy is the expression of an event-condition-action (ECA) rule. Policy management is distributed across the different functions of the architecture of autonomic managers as follows:

- Within the monitor function for extracting the relevant information. A part of the filtering and aggregation task can have been done in the monitoring feature.
- Within the analyse function for providing the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow the autonomic manager to learn about the IT environment, about the members and help predict future situations. The analyse function evaluates the different conditions that aims to update the global state of the system. This state and its changes are used as inputs in the condition part of the plan's rules.
- Within the plan function for providing the mechanisms that construct the actions needed to achieve goals and objectives and to enforce the VO's global policies. The plan function applies the adaptation policy and fires the rules acting on the system. Depending of the complexity of the operation, the number of steps, actions can be organized in a workflow process. In this case, the plan rules throws an action part that creates an instance of a process. The different interactions at each task can be held by others rules instead of human.
- Within the execute function by providing the mechanisms that control the execution of a plan.

2.5 Deployment Service Design

The goal of the deployment service is to install and configure applications in the VO as well as to remove applications. As discussed in section 2.2, this service deploys the components of the application according to (i) the resources provided by members and (ii) application constraints, such as locality constraints. The service relies on architecture descriptions, and supports component versioning and dynamic component update. In this section, we focus on the design of the deployment service.

Figure 7 presents the general architecture of the deployment system which is composed of four principal elements: (1) **the configuration and deployment description**, (2) **the targets**, (3) **the package repository**, and (4) **the deployment engine**.

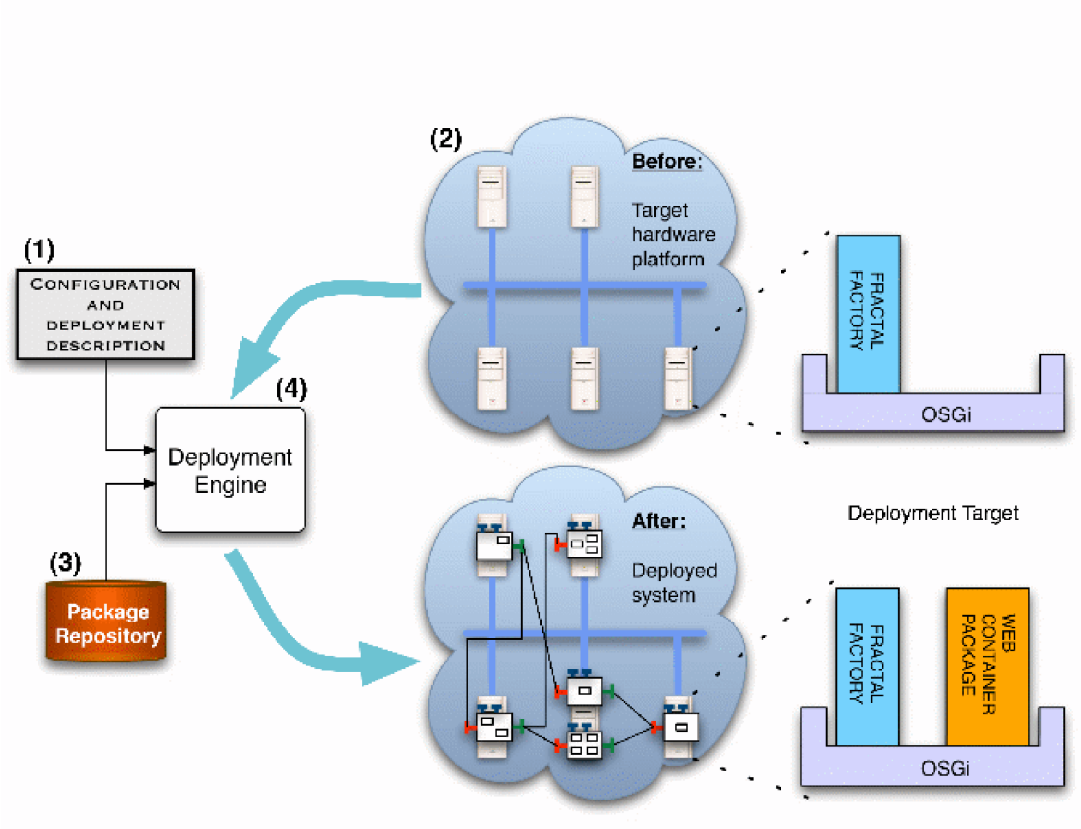


Figure 7 An overview of the deployment system

The configuration and deployment description specifies the applications that must be deployed within the VO. The targets represent the resources provided by members to the VO. The basic resources used by the deployment service are represented by the user's node. The package repository contains the bundles corresponding to the software to install. The deployment engine is the main deployment service. Below we describe in details all of those elements.

2.5.1 Configuration and deployment description

Configuration and deployment description is an input for the deployment system. It contains all the information needed by the deployment system to instantiate (deploy) a given application. A minimum set of such information is the following:

- Architecture of the application to be deployed i.e. components and their relation in terms of hierarchy and interconnections,
- Configuration of the components — values of their attributes,
- Placement information, i.e. placement constraints,
- Packaging information, i.e. in which software package is the code of a given component.

In the following we depict how to use our ADL to describe complex software configuration that can be deployed by the deployment service.

2.5.2 Complex software structure

The following Architecture Description Language (ADL) is an extension of the Fractal ADL. Therefore, it is XML-based and provides a static description of the complex system we want to deploy. Below is an example of a deployment file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE definition PUBLIC "-//objectweb.org//DTD Fractal ADL 2.0//EN"
  "classpath://fr/jade/service/deployer/adl/xml/jade.dtd">

<!-- ================================== -->
<!--        J2EE ARCHITECTURE           -->
<!-- ================================== -->
<definition name="J2EE">
  <interface name="service" role="server"
    signature="fr.jade.service.Service" />

  <!-- ================================== -->
  <!--                START               -->
  <!-- ================================== -->
  <component name="start"
    definition="fr.jade.resource.start.StartType">
    <virtual-node name="node1" />
  </component>

  <!-- ================================== -->
  <!--                APACHE              -->
  <!-- ================================== -->
  <component name="apache"
    definition="fr.jade.resource.j2ee.apache.ApacheResourceType">
    <attributes
     signature="fr.jade.fractal.api.control.GenericAttributeController">
      <attribute name="resourceName" value="apache" />
      <attribute name="dirLocal" value="/tmp/j2ee" />
      <attribute name="user" value="jlegrand" />
      <attribute name="group" value="jlegrand" />
      <attribute name="port" value="8081" />
      <attribute name="serverAdmin" value="julien.legrand@inrialpes.fr" />
      <attribute name="jkMounts" value="servlet" />
    </attributes>
    <virtual-node name="node1" />
    <packages>
      <package name="Apache HTTP server v1.3.29 (linux x86)" />
      <package name="Apache Wrapper" />
    </packages>
  </component>

  <!-- ================================== -->
  <!--                TOMCAT              -->
  <!-- ================================== -->
  <component name="tomcat"
    definition="fr.jade.resource.j2ee.tomcat.TomcatResourceType">
    <attributes
      signature="fr.jade.fractal.api.control.GenericAttributeController">
```

```
      <attribute name="resourceName" value="tomcat" />
      <attribute name="dirLocal" value="/tmp/j2ee" />
      <attribute name="javaHome" value="/usr/local/java/jdk1.5.0_05" />
      <attribute name="workerPort" value="8098" />
    </attributes>
    <virtual-node name="node2" />
    <packages>
      <package name="Tomcat (linux x86)" />
      <package name="Tomcat Wrapper" />
    </packages>
  </component>

  <!-- ================================== -->
  <!--                MYSQL               -->
  <!-- ================================== -->
  <component name="mysql"
    definition="fr.jade.resource.j2ee.mysql.MysqlResourceType">
    <attributes
      signature="fr.jade.fractal.api.control.GenericAttributeController">
      <attribute name="resourceName" value="mysql" />
      <attribute name="dirLocal" value="/tmp/j2ee" />
      <attribute name="user" value="jlegrand" />
    </attributes>
    <virtual-node name="node3" />
    <packages>
      <package name="MySql (linux x86)" />
      <package name="MySql Wrapper" />
    </packages>
  </component>

  <!-- ================================== -->
  <!--              BINDING               -->
  <!-- ================================== -->
  <binding client="this.service" server="start.service" />

  <binding client="apache.worker" server="tomcat.resource" />
  <binding client="tomcat.jdbc"   server="mysql.resource" />

  <binding client="start.rsrc_mysql"   server="mysql.resource" />
  <binding client="start.rsrc_tomcat" server="tomcat.resource" />
  <binding client="start.rsrc_apache"  server="apache.resource" />

  <virtual-node name="node1" />
</definition>
```

The description above defines a simple 3-tier architecture, which is built by Apache, Tomcat and MySQL servers. As specified by the virtual-node tag, each of the tiers should be deployed on a separate target machine. The virtual-node tag provides only collocation information, i.e. it does not provide information on the exact name/IP of the target machine, but only says which components should be placed together, and which should not. The only "dynamic" aspect of this description is the order in which the tiers are started.

MySQL needs to be started before Tomcat, which in turn needs to be launched before the Apache server. Since Fractal by default does not allow specifying the order in which components are started, Jade uses a specific component, called start, to achieve this goal. The start component launches all the components bound to it in an order equal to the one of bindings. Therefore, in the example above, starter will first launch MySQL, then Tomcat and finally Apache.

Information about component packages is provided through the package XML element. Each component can specify zero or more package elements, which are String package identifiers. Depending on the implementation of the package repository from which the packages are obtained, package identifiers can have different forms. At present we reuse the identifiers from OSGi Bundle Repository (OBR), as explained later. The rest of Deployment ADL is the standard elements found in Fractal ADL.

2.5.3 Architecture of the deployment service

Figure 8 presents the different layers of the deployment system, with well-defined interfaces between them.

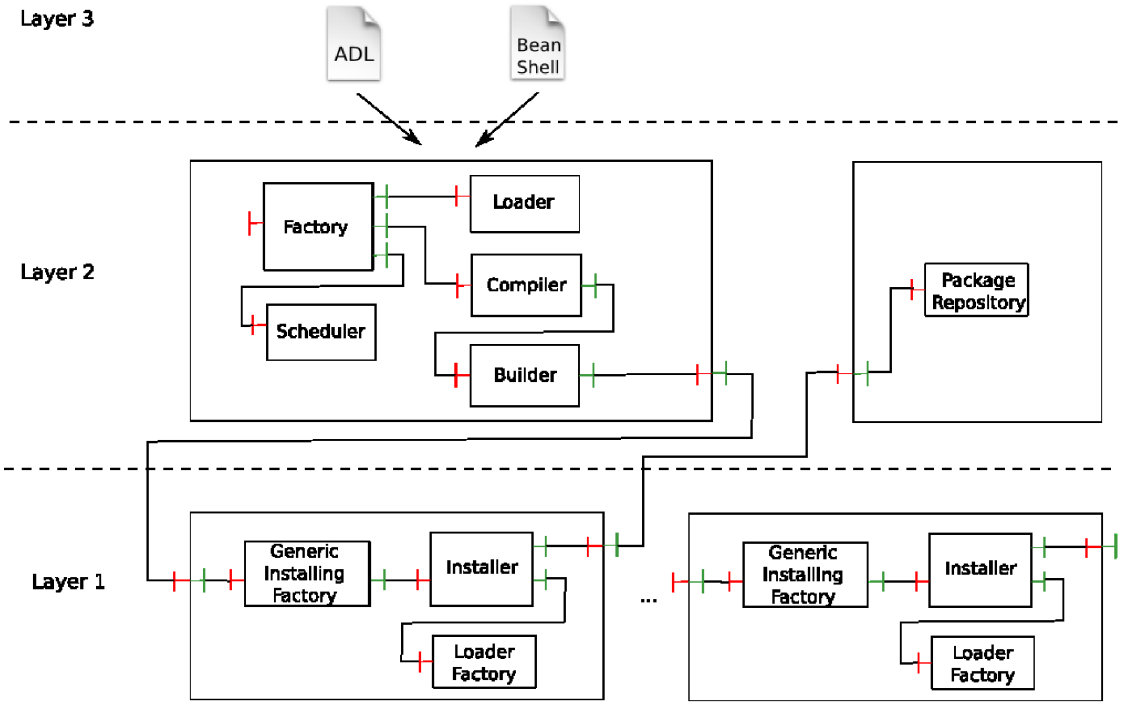


Figure 8 Architecture of the deployment system

Layer 3 consists of a (potentially distributed) application to be deployed.

Layer 2 is responsible for parsing the deployment descriptor; it interfaces with (i) the resource management service to allocate the required resources and (ii) the nodes (Layer 1) to deploy individual components. The main component of this layer is the deployment engine, which is detailed below.

Layer 1 is the local deployment environment for components. It provides support for component installation, instantiation, versioning, update, and removal. It exposes a local deployment API to the Layer 2, which hides the underlying installation environment. This API defines programming language independent abstractions, whose interactions are depicted in Figure 9. Currently, we are wrapping the OSGi environment behind these abstractions. This requires several modifications to the default Fractal component factory and some modifications to OSGi.

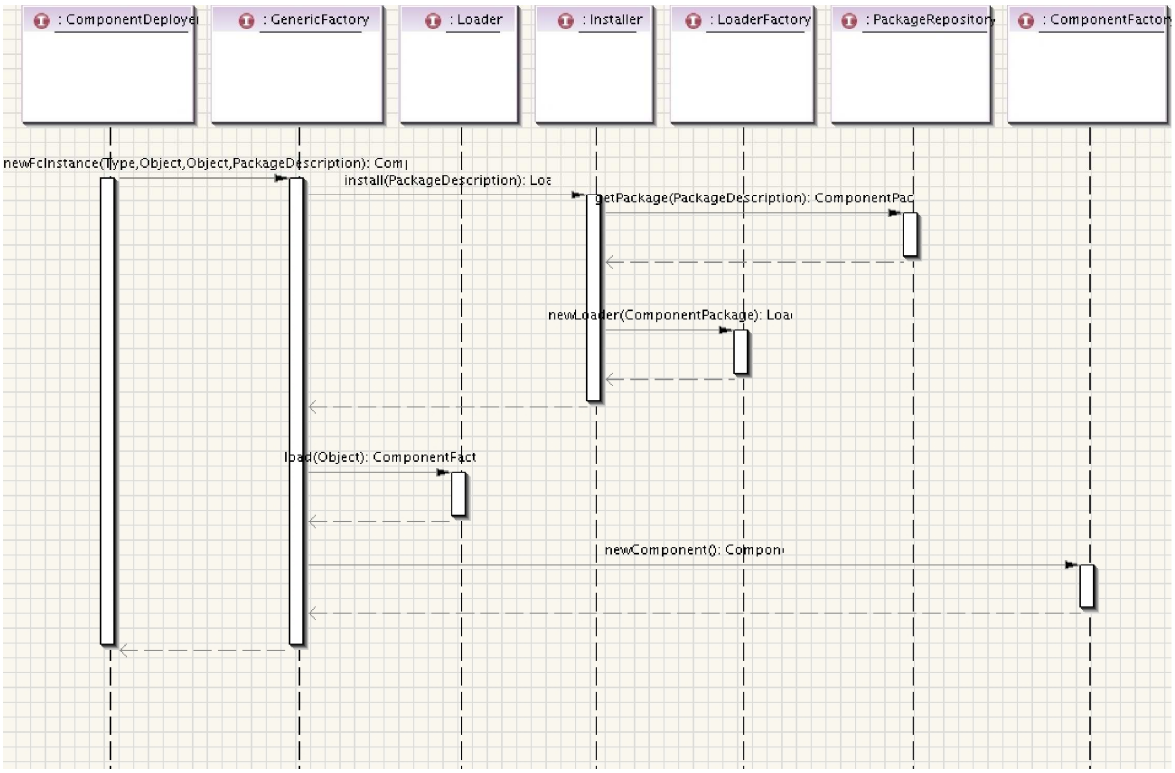


Figure 9 Interactions in Layer 1

The Jade deployment engine belongs to the Layer 2 in the deployment architecture. Given the Jade deployment description file as an input, the engine installs, instantiates, configures, and starts the components described by this file. In the current implementation, the engine is a largely modified Fractal ADL factory[4].

[4] `http://fractal.objectweb.org/current/doc/javadoc/fractal-adl/overview-summary.html`

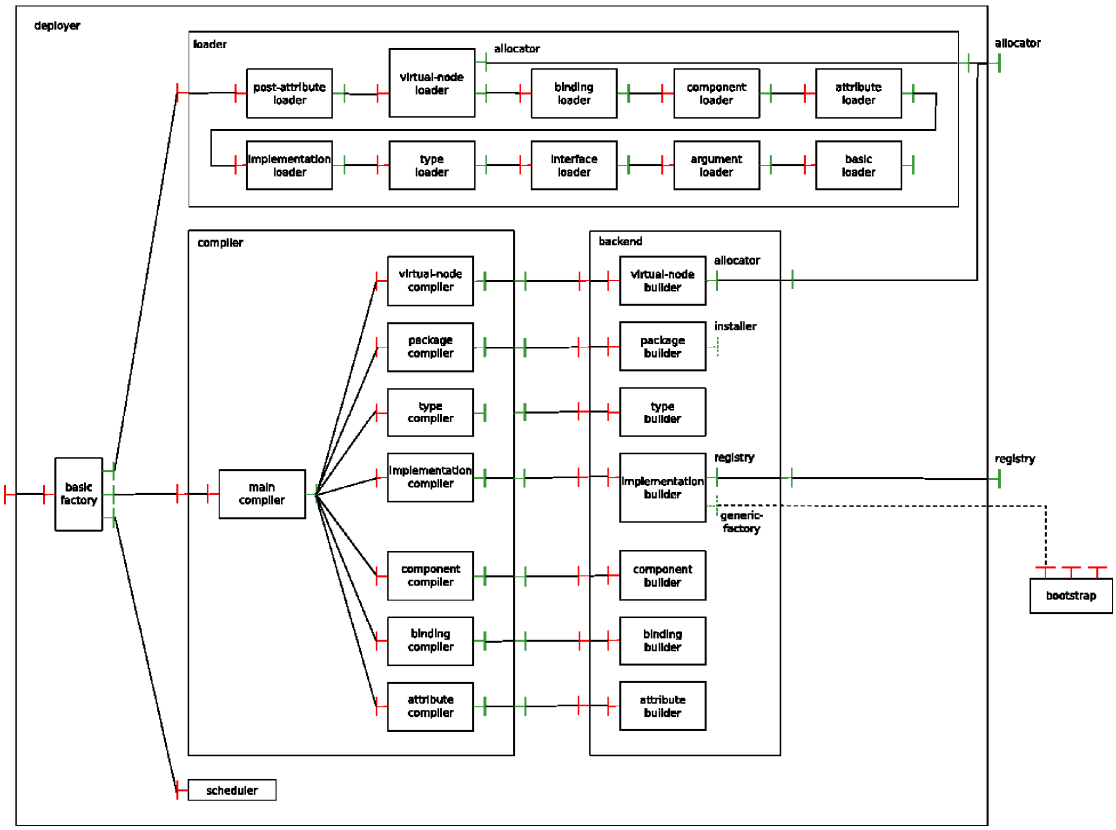


Figure 10 Jade deployment engine

The deployment engine is itself structured as a Fractal-based system that consists of three composite components, (1) the loader, (2) the compiler, and (3) the backend, each of them containing a set of primitive components. The loader component is responsible for verifying the correctness of the deployment file. The compiler component creates tasks that are executed by the backend component. The compiler component consists of several primitive components, which are executed in a top-down order; for example, the package compiler is executed before the type compiler. Each backend component implements a remote deployment command (AllocateResource, createComponent, setAttribute, Bind, addSubcomponent, Start).

As a first experiment, the deployment engine is not distributed; it resides entirely on a single machine. We plan to decentralize the engine using the component overlay services defined in the WP1 of the Grid4All project.

2.6 Conclusion and future work

We have presented in this document a general framework for the construction of virtual organizations as management domains. The framework comprises four main elements: a management domain structure, a notion of resource as controlled component, a set of core management services, and a notion of autonomic manager implementing automated management functions. Parts of this framework have already been implemented, as an outgrowth of previous work on the Jade autonomic management framework [6]. Further work will include: refining the framework to explain how it can make use of e.g. OGSA-compliant [19] and WSDM-compliant [9, 10, 55] services; extending the framework to include core security functions, such as role-based access control and trust management, probably along the lines of the TrustCom reference architecture [43]; specifying and implementing core management services in the context of a structured P2P overlay network, exploiting in particular the P2P programming interface defined as part of WP1 in the Grid4All project.

References

[1] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, M. Parashar, B. Khargharia, and S. Hariri. Automate: Enabling autonomic applications on the grid. In *5th Annual International Workshop on Active Middleware Services (AMS 2003)*. IEEE Computer Society, 2003.

[2] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A Gianoli, K. Lorentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. In *1st European Across Grids Conference*, 2003.

[3] N. Arshad. *A Planning-Based Approach to Failure Recovery in Distributed Systems*. PhD thesis, University of Colorado, USA, 2006.

[4] N. Arshad, D. Heimbigner, and A. Wolf. Deployment and Dynamic Reconfiguration Planning for Distributed Software Systems. In *15th IEEE Int. Conf. on Tools with Artifical Intelligence (ICTAI)*. IEEE, 2003.

[5] Don Batory. The LEAPS algorithm. Technical Report CS-TR-94-28, 1, 1994.

[6] S. Bouchenak, F. Boyer, S. Krakowiak, D. Hagimont, A. Mos, N. De Palma, V. Quéma, and J.B. Stefani. Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS 2005)*. IEEE Computer Society, 2005.

[7] E. Bruneton, T. Coupaye, M. Leclercq, V. Quema, and J.B. Stefani. The Fractal Component Model and its Support in Java. *Software - Practice and Experience*, 36(11-12), 2006.

[8] E. Bruneton, T. Coupaye, and J.-B. Stefani. The Fractal Component Model, *The ObjectWeb Consortium*, `http://www.objectweb.org`, 2004.

[9] V. Bullard and W. Vambenepe (eds). Web Services Distributed Management: Management Using Web Services (MUWS 1.1 part 1. Technical Report wsdm-muws1-1.1-spec-os-01, Oasis Consortium, 2006.

[10] V. Bullard and W. Vambenepe (eds). Web Services Distributed Management: Management Using Web Services (MUWS 1.1 part 2. Technical Report wsdm-muws2-1.1-spec-os-01, Oasis Consortium, 2006.

[11] D. Chadwick, A. Otenko, and E. Ball. Role-based Access Control with X.509 Attribute Certificates. *IEEE Internet Computing*, March-April, 2003.

[12] L. Cons and P. Poznanski. Pan: A High-Level Configuration Language. In *16th Systems Administration Conference (LISA)*. Usenix, 2002.

[13] Y. Diao, J.L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. A Control Theory Foundation for Self-Managing Computing Systems. *IEEE Journal on Selected Areas in Communications*, 23(12), 2005.

[14] Robert B. Doorenbos. *Production matching for large learning systems*. PhD thesis, Pittsburgh, PA, USA, 1995.

[15] Enabling Grids for E-sciencE (EGEE) project, 2005. URL: http://public.eu-egee.org/.

[16] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. In J. Mylopoulos and M. L. Brodie, editors, *Artificial Intelligence & Databases*, pages 547–557. Kaufmann Publishers, INC., San Mateo, CA, 1989.

[17] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *J. Comput. Sci. & Technology*, 21(4), 2006.

[18] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[19] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, Version 1.0. Technical Report GFD-I.030, Global Grid Forum (GGF), 2005.

[20] T. Friese, M. Smith, and B. Freisleben. Hot service deployment in an ad hoc grid environment. In *Second International Conference Service-Oriented Computing (ICSOC)*. ACM, 2004.

[21] D. Garlan, S.W. Cheng, A.C. Huang, B. R. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10), 2004.

[22] G. Goldszmidt and Y. Yemini. Distributed management by delegation. In *15th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 1995.

[23] P. Hasselmeyer, B. Koller, L. Schubert, and P. Wieder. Towards sla-supported resource management. In *Second International Conference High Performance Computing and Communications (HPCC)*, volume 4208 of *Lecture Notes in Computer Science*. Springer, 2006.

[24] G. E. Kaiser, J. J. Parekh, P. Gross, and G. Valetto. Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In *5th Annual International Workshop on Active Middleware Services (AMS 2003)*. IEEE Computer Society, 2003.

[25] H. Karlsen and B. Vinter. Vgrids as an implementation of virtual organizations in grid computing. In *15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2006)*. IEEE Computer Society, 2006.

[26] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer 36(1)*, 2003.

[27] F. Massaci, J. Mylopoulos, and N. Zannone. Hierarchical hippocratic databases with minimal disclosure for virtual organizations. *The VLDB Journal*, 15, 2006.

[28] A. Mowshowitz. Virtual organization. *Commun. ACM*, 40(9), 1997.

[29] H.A. Muller, L. O'Brien, M. Klein, and B. Wood. Autonomic Computing. Technical Report Technical Note CMU/SEI-2006-TN-006, Carnegie Mellon University – Software Engineering Institute, 2006.

[30] B. Nasser, A. Benzekri, R. Laborde, F. Grasset, and F. Barrère. Access control model for grid virtual organizations. In *Seventh International Conference on Enterprise Information Systems (ICEIS)*, 2005.

[31] B. Nasser, R. Laborde, A. Benzekri, F. Barrère, and M. Kamel. Dynamic creation of inter-organizational grid virtual organizations. In *First International Conference on e-Science and Grid Technologies (e-Science 2005)*. IEEE, 2005.

[32] M. Niinimaki, J. White, W. Som de Cerff, J. Hahkala, T. Niemi, and M. Pitkanen. Using Virtual Organizations Membership System with EDG's Grid Security and Database Access. In *15th Int. Workshop on Databases and Expert Systems (DEXA)*. IEEE, 2004.

[33] M. Parashar, H. Liu, Z. Li, V. Matossian, C. Schmidt, G. Zhang, and S. Hariri. Automate: Enabling autonomic applications on the grid. *Cluster Computing*, 9(2), 2006.

[34] J. J. Parekh, G. E. Kaiser, P. Gross, and G. Valetto. Retrofitting autonomic capabilities onto legacy systems. *Cluster Computing*, 9(2), 2006.

[35] J. Patel, W. T. Luke Teacy, N. R. Jennings, M. Luck, S. Chalmers, N. Oren, T. J. Norman, A. D. Preece, P. M. D. Gray, G. Shercliff, P. J. Stockreisser, J. Shao, W. A. Gray, N. J. Fiddian, and S. G. Thompson. Agent-based virtual organisations for the grid. In *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2005.

[36] Jess project, 2007. URL: http://www.jessrules.com/jess.

[37] Mandarax project, 2007. URL: http://mandarax.sourceforge.net/.

[38] L. Qi, H. Jin, I. Foster, and J. Gawor. Hand: Highly available dynamic deployment infrastructure for globus toolkit 4. In *15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2007)*. IEEE Computer Society, 2007.

[39] K. Ravindran and A. Mowshowitz. 'virtual organization': a Service-level Resource Management Framework for Distributed Network Infrastructures. In *Globecom '03*. IEEE, 2003.

[40] P. Robinson, Y. Karabulut, and J. Haller. Dynamic virtual organization management for service oriented enterprise applications. In *1st International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2005.

[41] G.D. Rodosek, H.G. Hegering, and B. Stiller. Dynamic Virtual Organizations as Enablers for Managed Invisible Grids. Technical Report 2005.09, Dep. of Informatics, University of Zurich, Switzerland, 2005.

[42] G.D. Rodosek, H.G. Hegering, and B. Stiller. Dynamic Virtual Organizations as Enablers for Managed Invisible Grids. In *10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006.

[43] L. Schubert, M. Wilson, J. Haller, A. Arenas, A. Svirkas, P. Giambiagi, J. Doser, E. Lupu, N. Tuptuk, and L. Martino. TrustCom Reference Architecture. Technical Report Deliverable D09, TrustCom Project IST-2002-01945, 2005.

[44] DataGrid WP07 Network Services. Security Requirements and Testbed 1 Security Implementation. Technical Report DataGrid-07-D7.5-0111-4-0, DataGrid Project IST-2001-25182, 2002.

[45] B. Shan, Y. Han, and H. Wang. Enabling virtual organizations with an agent-mediated service framework. In *10th International Conference on CSCW in Design, CSCWD 2006*. IEEE, 2006.

[46] D. Shands, J. Jacobs, R. Yee, and E. John Sebes. Secure virtual enclaves: Supporting coalition use of distributed application technologies. *ACM Trans. Inf. Syst. Secur.*, 4(2), 2001.

[47] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[48] R. Sinnott, D. Chadwick, J. Koetsier, O. Otenko, J. Watt, and T. Nguyen. Supporting Decentralized, Security focused Dynamic Virtual Organizations across the Grid. In *2nd IEEE Int. Conf. on e-Science and Grid Computing (e-Science)*. IEEE, 2006.

[49] M. Sloman and K. Twidle. Domains: A Framework for Structuring Management Policy. In Morris Sloman, editor, *Network and Distributed Systems Management*, Chapter 16. Addison-Wesley, 1994.

[50] Morris Sloman. Policy driven management for distributed systems. *J. Network Syst. Management*, 2(4), 1994.

[51] G. Valetto. *Orchestrating the Dynamic Adaptation of Distributed Software with Process Technology*. PhD thesis, Columbia University, USA, 2004.

[52] G. Valetto, G. Kaiser, and D. Phung. A Uniform Programming Abstraction for Effecting Autonomic Adaptation onto Software Systems. In *2nd Int. Conf. on Autonomic Computing (ICAC '05)*, 2005.

[53] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Comput. Surv.*, 38(1), 2006.

[54] G. Wasson and M. Humphrey. Toward Explicit Policy Management for Virtual Organizations. In *4th Int. Workshop on Policies for Distributed Systems and Networks (POLICY)*. IEEE, 2003.

[55] K. Wilson and I. Sedukhin (eds). Web Services Distributed Management: Management of Web Services (MOWS 1.1. Technical Report wsdm-mows-1.1-spec-os-01, Oasis Consortium, 2006.

[56] L. J. Winton. A Simple Virtual Organisation Model and Practical Implementation. In *ACSW Frontiers – Australasian Workshop on Grid Computing and e-Research (AusGrid 2005)*, volume 44 of *CRPIT*. Australian Computer Society, 2005.

[57] Y. Zuo and B. Panda. Component Based Trust Management in the Context of a Virtual Organization. In *ACM Symposium on Applied Computing (SAC)*, 2005.

3. Grid4All Market – requirements, state of art, architecture

Grids are distributed computing infrastructures that enable sharing of resources. Grid4All focuses on consumer and neighbour-hood Grids, as a peer and small-organization based counter-part to current computational and data Grids that focus on large organizations. The proliferation of affordable yet powerful desk-top computers capable of an appreciable computing and storage capacity backed by the growth of high-speed networks has made Internet computing pervasive.

Chapter 1 has addressed the creation and management of spontaneous and ad-hoc virtual organisations that may be created any actor within the Internet – schools, domestic users, internet clubs, and even small enterprises. Deliverables 4.1 and 4.2 from WP4 describe applications and provide motivating use scenarios that show the need to provide management support to operate virtual organisations in an open and dynamic environment.

The management concern addressed in this chapter is that of on-demand resource allocation to virtual organisations. Internet-connected computer systems possess surplus bandwidth, storage, and computational resources. Resources are pervasive and inexpensive, and we would like to render their provision dependable and consistent and satisfy the resource needs of virtual organisations. This is motivated by the need to permit small organisations to access managed IT resources without needing to increase their investment in capital and operational expenses.

The first question to address is why Internet users should provide their resources to others? Projects such as [SETI] and successors rely on the sensibility of Internet resource owners towards scientific and other applications of benefit to society. More recently peer-to-peer applications and particularly file sharing networks propose co-operative usage models where peers trade resources between themselves – this rupture is mainly thwarts commercial service providers. It relies on the members of collaborating communities to mutually provision resources.

On the industrial side, utility computing is establishing itself as a business model whereby computer resources are provided on an on-demand basis. Even large enterprises are moving to this model as a means to realize economies in investment on IT infrastructure and in technical expertise needed to maintain and manage such infrastructures. An abstraction adopted in this context is to consider provisioning of computational resources as a service. [FK02] defines computer and storage resources and in general any Grid application, as a service that may be accessed through the network. The convergence of Grid computing with Web Services has taken this even further. Grid services are considered as Web Services that are accessible over the network.

Virtual organisations are formed by users that have common goals and need to collaborate to achieve their technical or business objectives. Such virtual organisations require computational resources to satisfy the needs of members and applications. These requirements may typically vary over the life-time of the virtual organisations. Allocation of computational resources on-demand from the Internet will permit the participating organisations to reduce their internal investments. However relying on voluntary donations of resources is not viable. Approaches to resource sharing that are based on co-operation, such as within peer-to-peer networks has shown its limitation; tension exists between individual rationality and collective welfare [FECH05] as shown by the free-riding phenomena in P2P systems that threatens the viability of these systems. An alternative approach is that of establishing Grid resource markets. Markets have proven their ability to allocate resources efficiently; markets allocate resources to who needs them the most and importantly provide mechanisms through which the need may be correctly elicited and quantified. Markets also promote incentives to resource owners to provide or trade their resources. In computing environments, resources that are traded are processing time, storage, or applications, where examples of the last may be an efficient codec, a parallelizing compiler. In this work package (WP2) we advocate markets to allocate resources to satisfy the needs of virtual organisations.

3.1 Motivation for market based Grid resource management

Virtual organisations are coordinated groups of individuals and institutions that have common interests and objectives. They share, on the basis of some policies, a set of resources. Participating entities are self-interested but realize the potential benefits from pooling resources. Members may be geographically distributed and can access to the resources any time they are allowed. Organisations entering in collaboration, such as for a long running scientific project, create a planned and pooled infrastructure. Local administrators are responsible to manage the local resources and shared services.

In current Grid computing, the term virtual organisation mainly refers to security domains. Chapter 1 extends this definition. A virtual organisation is put in par with Grid – it is the ambition to build an organisation with many functional properties of a real organisation. Concerning the object of this task, we are interested in how resources are accessed in such organisations – resources are transformed as a product or service.

Service oriented grid computing virtualizes resources and by this significantly increases the versatility of the Grid. Moving from the context of large enterprises and academic institutions, within Grid4All we envisage formation and the run-time management of VOs – targeting even small organisations such as schools or domestic providers and consumers of resources. These structures should be able to harness unused networked resources on demand and access Grid resources as any other service on the Internet. At creation the VOs operate with the minimum needed computing resources contributed by the member organisations. At run time resources are acquired on-demand from resource providers, thus enabling these virtual organisations to perform computational tasks using *leased* resources. This extends the notion of utility computing to virtual organisations.

The typical scenario that is envisaged in Grid4All is a set of schools that join together to form a VO: some of their students jointly participate in a large science fair and collaborate to design and develop the experiments, the necessary scientific software. At a later stage they need to execute the software and gather results. Here, computer resources should be accessible in analogy to the power grid in a plug-and-play manner. The school VO should seamlessly be extended to use virtualized compute resources on the Internet. In order to describe the needs and behaviour of the system, a development team – here at the school, will define the system components and applications needed to achieve the required functionality. Chapter 1 has proposed an Architecture Description Language by which a formal description of the system is specified based on which management services deploy and configure applications. Dynamic application and service deployment on newly acquired resources is an essential feature needed to enable utility computing.

Logically, the organisation that we envisage is that the computational resources on the network are *partitioned* across virtual organisations. Allocated resources belong to one or other of the virtual organisations whose configuration adapts dynamically to change in load. VOs allocate resources from resource markets when there is a need. This is similar to recent approaches in resource management and provisioning that takes roots from utility computing. [CHI03] presents *Cluster-On-Demand (COD),* a resource management system that allocates servers from a common pool of clustered resources to multiple virtual clusters. Each virtual cluster has independently configured software environments, names spaces, user accounts, and networked storage volumes. Resources are allocated to virtual clusters dynamically on-demand based on sharing policies and adaptive provisioning -- the node allotments to virtual clusters change according to competing demands and resource availability.

Shirako [ICH06], a successor of *Cluster-On-Demand (COD)* extends these principles to resources on a wide-area network, through brokered leasing of resources. Resources contributed by autonomous sites – resource providers -- are pooled and managed by lease *brokers.* The clients of the leasing service are *guest* applications – each application is managed by a *Service* manager that is responsible to negotiate the leases with the brokers on behalf of the guest application.

These architectures show a "*two-level"* management of resources: each virtual cluster determines internally the mapping of its resources to its applications based on internal objectives and policies. The choice of the internal scheduler and work load management system depends on the type of applications. At the higher level, brokers arbitrate the allocation of global resources to each of these virtual clusters. Virtual clusters

respond to load surges by leasing additional resources. It may also react to external resource contentions by accepting reduced service levels and hence free resources.

We have adopted this two-level management for Grid4All virtual organisations. From clusters of [CHI03] we transpose to a heterogeneous network of computers on the Internet and from virtual clusters to virtual organisations. VOs adapt to changing conditions such as load surges or failures by *leasing* resources.

From whom may resources be leased or allocated? Major firms in the computing industry such as IBM, HP, and SUN Microsystems are focusing on agility and flexibility of computing resources and gearing their versions of on-demand computing and IT outsourcing solutions. Utility providers i.e., operators such as Amazon sell raw computing power at fixed prices. Such operators have invested on resources such as cluster farms to furnish utility computing services. But, there are also a large number of *idling* resources on the Internet. Could we *use* these resources to offer a utility computing service? Incentives to contribute resources have been driven by public good or collaborative advantage. This is manifested in test-beds such as [SETI]. These cannot be generalized since the motivations of contributors are not amenable to generalized models of sharing; so what other incentives for these resource owners and how to arbitrate their allocation. Can a market based approach provide a solution?

[DWP86] defines a *market* as a context in which the sale and purchase of goods and services take place. [DR92] suggests a definition by which market is a medium of exchanges between buyers and sellers. A *good* is the economic abstraction for a thing that imparts utility to its possessor or recipient. A market transaction takes place when all parties perceive that their own utility will not decrease by their participation, relative to not participating. We consider that [TUCKER98], "a market is a medium in which autonomous agents exchange goods under the guidance of price in order to maximize their own utility".

Markets rely on consumers to set a value on the resources that they want and based on these values provide optimal allocations. Consumers are endowed with a budget and seek a quantity of resource that maximizes their internal utility, given the current prices. Trade occurs at a clearing price that balances supply and demand – such allocations are also *economically efficient,* that is, no reallocation can make one better off without making another worse off. By Adam Smith's *invisible hand* [SMI79], perfect competition where buyers and sellers act independently and selfishly, channel scarce resources to economically efficient users. The invisible hand that guides buyers and sellers is the market price – users buy until their marginal benefit equals price. Pricing of resources becomes the regulatory mechanism that addresses fluctuations in supply and demand. Price evolves according to market dynamics and is indicator of demand, higher the demand, higher the price. This information is used by agents to decide on the quantity of resources that they acquire and when they acquire them.

[BAG00] have formulated a set of questions whose answers help to decide if an economy driven resource management system is warranted. We answer some of these questions within the context of Grid4All.

Who are the resource providers and the resource consumers?

Majority of the expected resource providers and consumers are domestic owners of computers. Small enterprises and organizations such as local schools may also provide resources, but more often consume resources. Most of these are now connected to the Internet through broad-band access networks.

These consumers do not in general expect free resources, but low cost access to resources. Such consumers seek to reduce their initial IT investment and also running costs. They may be prepared to sacrifice on quality of service, nevertheless do demand an acceptable level of consistency and dependability.

What motivates one to contribute their resource to the Grid?

Not all VOs search for extra-terrestrial intelligence! Resource owners will contribute resources for a correct compensation. Compensations may be reductions on tariffs of broad-band access, or perks on other services that they access through their Internet operator – such as video-on-demand, television over Internet. Within Grid4All we expect that most members are more self-interested rather than altruist and expect some benefits in return to the resources that they contribute.

How can users solve there problems within a minimum cost?

In general, users can reduce their costs by reducing their investment, both capital and operational expenditure, by considering computational resources as a Utility. Users adjust to market conditions; for example relax dead-lines if this enables them to acquire cheaper resources.

Is access cost the same for peak and off-peak hours? Accessorily what to do when there are more requests than available resources?

In the case of excess demand, requests cannot be prioritized – how to give priorities in the Internet? However costs are dissuasive. Basing prices on supply and demand will necessarily motivate rational users to adjust their demand, thus reducing congestion.

How can resource owners maximize profit?

They can maximize profit by contributing resources to who value them most, that is, to who pay the most.

Users create and form VOs so as to reach a set of objectives. VOs execute applications and services that allow the creators to achieve their goals. The principle issue is that of arbitration when there is excess demand. Straightforward allocation policies are essentially based on priorities perhaps subject to constraints on quota of utilisation. If self-interested participants are free to set their own priorities then they will each specify the maximum priority since they do not have the incentive to do the contrary. Moreover this approach does not provide incentives to owners to share their resources unless of course the each provider is also a consumer.

Thus market-based resource management is expected to be the most satisfactory when participants are dominantly self-interested. Resource providers or owners have incentives to share their resources if they are adequately compensated. Pricing of resources establishes a common scale of value across various resources and resources are allocated to those who value them the most.

3.2 State of art of market based resource management systems

3.2.1 Related work on market based resource allocation

Market based systems are dominantly classified by the market models that they use. [BY06] provides a detailed taxonomy differentiating the market models that may be applied in market-based systems.

The phrase *market mechanism* is encountered in connection with problems of distributed resource allocation. *Mechanism* in the context of markets refers to a structure of economic organization that helps to shape outcomes. Intuitively [NiRo99] a mechanism solves a problem by assuring that the required allocation occurs when agents choose their strategies to maximize their own utility. A mechanism also needs to ensure that the agents reported utilities are compatible with the algorithm implementing the mechanism. Economic mechanisms propose a procedure by which a set of resources may be distributed amongst the different participants and a scheme for pricing of the traded resources. The allocation is constrained by the preferences of the participants expressed in monetary terms.

The annex 3.10.1 presents a classification of market models.

Auction markets for single type of resource

- *SPAWN* [WALD92] was designed to tap in unused and wasted cycles in networked servers. Each participating server runs an auction process to trade the CPU time in fixed time-slices. Spawn uses a sealed bid second-price auction, known also as the Vickrey auction. Vickrey auction is incentive compatibility, i.e. the best strategy that the bidders may practise is to reveal their true valuations. This system is not generalized to multiple resources and multiple resource units – considering time-slices as a resource unit; this implies an auction for each time-slice.

- [PLA06] presents a trading platform for storage services. The platform implements a centralized storage exchange that implements a double-auction; the exchange accepts sealed offers from providers and consumers and periodically allocates trades by employing an algorithm that maximises surplus, that is, the difference between the consumer's price and the seller's cost. Double auctions are adapted to trading of a single type of homogeneous resources. These have the benefit of reducing communication costs (single bids), and with suitable pricing policies are also incentive compatible.

Auction markets for multiple types of resources

Combinatorial auction model has received a lot of attention in recent years; to address trading multiple resource types in bundles; this has two implications: (i) prices are expressed for bundles and (ii) a bundle if allocated should be completely satisfied.

- [CHUN04] presents a resource discovery and allocation system where users may express preferences using a bidding language that supports XOR bids implying that at most one of the preferences is to be allocated. Multiple resources may be requested to a central auction server that clears periodically. Resources are requested for fixed time durations and users may specify the time ranges. A greedy algorithm clears the combinatorial auction. This algorithm privileges execution time over efficiency of allocation – bids are ordered by decreasing values where the value is obtained by dividing the bid price by the product of total number of resources and the duration of request.
- [SCH06] presents an iterative combinatorial auction that maximizes seller revenues. Bids are presented as a two-dimensional matrix; one dimension represents the time in fixed time slots and the other dimension the resources (CPU, Disk, and network). The auction server executes periodically and invites bids from the participants. Shadow prices are calculated for individual items (resource) and the buyers are requested to iterate on their bids based on the current estimation of prices. The clearing algorithm is implemented as a linear program optimizing the revenue. Prices are calculated using the approach presented within [KWAS05] – prices are the dual solution to the primal LP.
- [SCH05] presents a multi-attribute combinatorial auction that maximizes the surplus – the difference between the buyer's price and seller's cost. Resources are traded in fixed time-slots and buyers send XOR bids specifying the quality and the number of time-slots within a specified time range. The Vickrey pricing policy is applied to provide incentive compatibility. Simulation results show that the allocation problem is computationally demanding but feasible in the case where the number of participants and bids are reasonably small.

Even though the above approaches indicate the computational complexity of combinatorial auctions, they nevertheless are extremely important demonstrators. Combinatorial auctions are important mechanisms for Grid resource markets since typically Grid applications need to allocate resources in bundles and furthermore Grid resource requests represent complementarities.

Commodity markets

- [WOLSKI01] presents *G-Commerce* which provides a framework for trading computational and storage resources through commodity markets. Commodity markets allow applications to treat disparate resources as inter-changeable. Commodity market operates through an adjustment scheme that approximates the prices to equilibrium making the assumption that the commodities are inter-related.

Bargaining

- [CBF02] presents *Stanford Peers* which is a peer-to-peer storage system where each peer acts both as a consumer and provider of storage. It can be considered to be a cooperative sharing environment that *swaps* resources between peers. The amount of storage is considered as the 'price' in the system. At the simplest level peers swap equal amounts of storage. It uses auction like methods to implement swapping – consumers call for auctions and wait for proposals. A number of policies are proposed for both bidding and bid selection. One policy is that the amount of space that is bid by a given provider site is equal to the amount of space provided by the consumer site to this provider.

Proportional sharing markets for divisible resources

Market based *proportional sharing* models are one of the most popular approach in problem-solving environments. Basically this approach consists of allocating to users a percentage of the resource that is proportional to the amount of the bid submitted by the user. This may be considered as a fair model of allocation and is typically employed in cooperative environments employed in systems where resources are considered as divisible.

- [FLZ05] presents *Tycoon* is a system designed for time-sharing networked nodes such as in PlanetLab; an environment where users of resources are also providers of resources. Resource nodes execute an auction process to which users may send their bids. *Tycoon* implements a proportional sharing [KEL97] auction where each user is attributed a capacity proportional to its bid. Users are price-anticipating in that the ratio that they receive is a proportion of their bid over the sum of all bids for a given resource -- users may anticipate the effect of their bids on the clearing price. Each user has a utility function; a weighted sum of the resource fraction that it receives from each node. Users bid to those nodes that maximize their utility. This system is intrinsically decentralized as the maximization of utility is done locally at each user. This system is appropriate for divisible usage of CPU, an assumption that may not be acceptable to a wide range of applications.

3.2.2 Related work on architecture of computational resource markets

Computational economic frameworks have adapted the paradigm of societal economies to allocate resources in distributed systems. Current state-of-art projects are mainly proof-of-concept demonstrators that show the potential of this approach to allocate computational resources. [BAV04] summarizes some important benefits of a Grid Economy:

- ✓ It helps in realizing large-scale Grids as it offers incentives for resource owners to contribute their resources,
- ✓ It regulates supply and demand and resolves conflicts,
- ✓ It helps to build scalable systems since the decision making process may be distributed to all users and resource owners,
- ✓ It places the power in the hands of both resource owners and users – they make their own decisions to maximize the utility gained and profit.

Efforts in standardisation for Grid Economics were started by Grid Economy Services Architecture [GESA03] working group of the Global Grid Forum that specified a set of mechanisms (protocols and service interfaces) to enable Grid Economies. The proposed architecture is based on the now out-dated Open Grid Services Infrastructure (OGSI). The GESA consisted of service interfaces and definitions for three key elements within an economic architecture (i) account resource usage (ii) present the conditions of access to a Grid Service through a notion of Chargeable Grid Service and (iii) and finally provide a payment infrastructure through the Grid Banking Service.

There has been a considerable progress since these early efforts; two representative architectures are summarized below.

GRACE

[BAV04] is one of the first to propose a comprehensive architecture to promote the Economy based Grid. It has established some requisites for any economy-based Grid system:

- ✓ Information and market directory for publishing Grid entities
- ✓ Models for establishing the value of resources
- ✓ Resource pricing schemes
- ✓ Economic models and negotiation protocols
- ✓ Mediators to act as a regulatory agency to establish resource values and currency standards
- ✓ Accounting, Billing, and Payment mechanisms

Figure 1 presents the functional Grid Architecture for Computational Economy (GRACE) [BV04] which has come to be considered the reference architecture within some Grid communities.

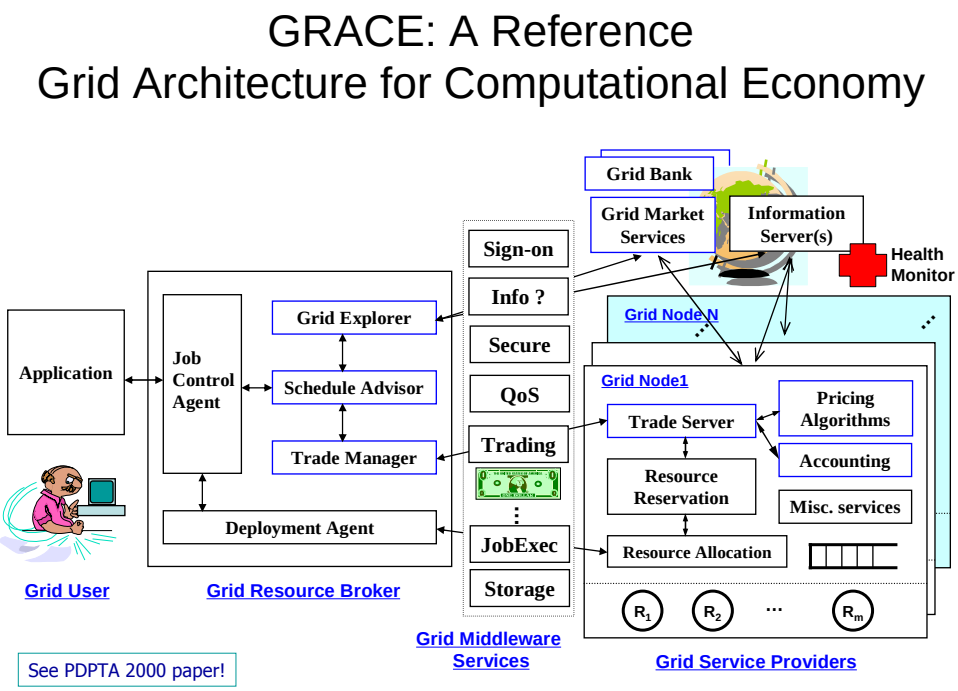


Figure 11 Reference Economic Grid Architecture (courtesy Buyya)

The Trade Manager represents end applications that require resources and is responsible to interact with trade servers and negotiate access to the resources. Its objective is to minimize the cost to applications. The Trade Server is an agent that executes on behalf of a resource owner and sells access to resources. Its objective is to maximize the profit to its owners. Grace defines the rules and formats for exchanging negotiation messages between these two agents through the Trading Protocol. Trade Servers are designed to allow pluggable pricing policies; they are accessed through open trading APIs that hide the choice of the underlying pricing schemes.

Albeit being a comprehensive architecture, there are nevertheless some points to note:

- The system is based on bi-lateral negotiations between consumer and provider. This leads to problems of exposure when resources may need to be allocated from multiple providers.
- The software architecture of the Trade Manager and the Trade Server is not open.
    - It is not clear how amenable the Trade Server is to supporting multiple market mechanisms even at design.
    - Reciprocally, the assumption that a few defined utility/demand functions may be valid for a large range of application.
- Adaptation of Trade Managers in face of different negotiation protocols is inadequately supported.
- Resource providers are assumed to be job schedulers. It is not trivial to extend to different types of resources.
- Finally, it is not clear whether this architecture can be easily extended to incorporate new functionality. Commercialization of Grid resources needs to incorporate aspects such as service level contract management, reputation etc. that are mandatory within economic architectures

OCEAN

[PP03] presents an open software infrastructure to automate trading of heterogeneous computing resources on the Internet. In contrast to GRACE, this infrastructure aims at completely decentralized architecture facilitated through a peer-to-peer search protocol that can quickly find suitable matches among large number of providers and consumers. The multi-layered OCEAN infrastructure is architected to be portable over various cluster and Grid infrastructures.

The economic services of the OCEAN architecture are encapsulated within a trading services layer that provides the following functionalities:

- The Auctioning component focuses on locating suitable trading partner for nodes in the OCEAN network by executing a distributed double-auction mechanism. This component provides a trading proposal API that allows buyers and sellers to generate XML documents describing the resource requirements and availability.
- The negotiation component provides the means for traders to agree in the terms of a contract document based on XML schemas. The negotiator bases its search for peers based on the policies set by the trader layer. This layer currently does not support simultaneous negotiation among multiple nodes.
- The accounting component logs all successful negotiations and the resulting transactions at a centralized accounting system.

The main feature of the OCEAN architecture is the peer-to-peer based directory service and matching network that determines a candidate list of peer nodes for each node. This list represents the potential negotiation partners and is based on status information such as the availability, inter-node bandwidth. This layer provides query mechanisms to obtain the configuration information of peer resources. The main limitation is that of the lack of adaptability of the architecture to support multiple auction mechanisms. Moreover a syntactic resource description framework is restrictive to implement open market places.

CATNETS

[EYM03] presents the middleware architecture that purports to provide to build economic and market based resource management systems. It uses the notion of Application Layer Networks (ALN) that integrates different overlay approaches – Grid or peer-to-peer systems, on top of the Internet. The architecture is driven by needs for (i) scalability (ii) self-organization and self-adaptation, (iii) openness. Scale is addressed by restricting requirements for the maintenance of global state – this is infeasible on large ALNs that could contain thousands of nodes. Self-organization is addressed through an approach based completely on decentralized negotiations; this avoids centralized co-ordination and allows peers to continue to obtain services even on conditions of failure and volatility that is to be expected in large-scale peer-to-peer networks. Finally an open architecture is important for Internet scale resource management systems; this is addressed through a layered software architecture that isolates economic agents from the underlying ALN, by providing support for pluggable mechanisms and strategies, and by taking an approach based on specification of messages using XML – this last renders language and platform independence.

Discussion

The important lesson learnt from each of these architectures is the requirement for open APIs and layered software architecture. OCEAN and CATNETS focus on a completely decentralized system based on direct negotiations between peers. This nevertheless renders allocation of bundled resources difficult. The GRACE architecture is more adapted to Grid systems formed of a small number of nodes – where each node represents a large clustered system. OCEAN lacks support for the implementation of multiple auction mechanisms, and in fact does not address this need. An open market place for Internet level Grids needs to address both in terms of development and run-time co-habitation of multiple market mechanisms. Even though CATNETS and GRACE demonstrate the need to provide support for multiple market mechanisms, the aspect of inter-operability of agents in the face of multiple market negotiation protocols is not addressed.

3.2.3 Related work on currency and payment

Currency serves as an element to assign value or generic capacity to consumers, to facilitate and delegate actions/uses, to account for usage of services and resources, and to support the regulation of a system by enabling or limiting the generic capacity of consumers in respect to the capacity of providers and also limiting the storage of value.

Virtual currency has arisen as a necessity to apply economic policies in the scope of the regulation in distributed systems. Every system based on economic models has defined some type of currency in an explicit or implicit way. The interesting properties currencies have are:

- **Common unit of accounting:** A common unit to value goods traded in a system.
- **Purchase power:** A mechanism to prioritize the use of different goods based on their necessities.

The design of a currency system is bound by trade-offs between security, scalability and availability. This section defines a classification and the design of some major currency systems, both from economic aspects and that of system aspects such as security.

Multiplicity of currencies: An economic system may be populated either by a single currency or by a multiplicity of currencies. Typically a currency has a region within which it is dominant. For example, in Europe there is the Euro; in the United States is the Dollar, etc.

Electronic payment systems such as PayPal or e-Gold implement exchanges between different real monies. Most of the distributed computing systems using currency management implement a single monetary unit as exemplified by systems such as Ppay, Tycoon, Karma, SecondLife, and World of Warcraft.

Currency Market: Currency market is the entity by which one currency is traded against another at a set price or rate, called an exchange rate, required when there is more than one circulating currency. Two issues addressed in Grid4All are (i) an exchange mechanism and (ii) control on circulation of virtual currency.

Currency representation: Two basic abstractions are used to represent the value of currency: account balance based and token based. In *account balance based systems*, peers maintain an account with a bank. Payments are effected as bank transactions between accounts. Centralized implementations [Lai05], [Rege98] face issues of load and availability. Decentralized implementations [Vish03], [Garc04], [Haus05] need to address presence of malicious users.

Token based systems issue tokens to peers that they use to obtain services from other peers. Clients transfer tokens to suppliers of service. This is similar to cash and similar to cash counterfeiting needs to be detected. PPay [Yang03] based on a DHT implementation allows each peer to manage its tokens. [Lieb04] proposes a secure protocol to perform token transfer. [Chau90, Okam92, NuNg01, Brand93] maintain the privacy and security of the users involved in a transaction at the cost of complex implementations. Token based systems need to implement algorithms to avoid (or detect) malicious behaviour such as double-spending, counterfeiting

Currency storage: A currency system may store currency either locally or remotely. Remote storage may be centralized or distributed. Token based systems such as [Fu03, Chau90] allow each user is in charge of storing his currencies (tokens) in a local persistent device like a wallet. Central remote storage based systems introduce the notion a bank that regulates behaviors of users. Tycoon [Lai05], PopCorn Market [Rege98] and GridBank [Barm02] follow this approach. They focus on heuristics to reduce the number of transactions that require the intervention of the bank thus reducing the computational load of the bank.

In *remote distributed storage* based systems, transaction and account management is distributed between the participants. PeerMint [Haus05] and Karma [Vish03, Garc04] implement a distributed bank where each peer (account holder) is responsible for the account of another peer (account owner). Accounts may be replicated to guarantee availability.

Bank intervention degree: Currency transaction or payment protocols are classified based on the degree of intervention of the bank. *Off-line payment protocols* execute payments directly between customer and provider and the role of the bank is to issue currency. Malicious behavior such as double-spending is not prevented, but detected lazily (after fraud has been committed). [Okam92, MuNg01, Brand93] implement this solution at the cost of delayed detection of multiple spending. *On-line payment protocols* [Garc04, Haus05] are safer since each transaction involves the bank, a trusted entity that verifies correct behavior.

Ordering of transactions: Atomicity may not be guaranteed when transacting Grid goods; one of the two operations, *pay* and *acquire* will always precede the other. Nevertheless participants should be able to agree on the protocol: *pay before acquire, pay during acquire, pay after acquire. Paying before acquiring* the goods does not guarantee the customer and *paying after acquiring the goods* does not guarantee the supplier, requiring mechanisms that allow providers and consumers to reclaim their dues. *Paying during acquiring the goods* is a compromise. Payments need to be done continuously and regularly during the usage of resources. Either party may terminate the contract in case of default in payment or default in provision of service. This mechanism is adapted when resources are used over a long period. GridBank [Barm02] proposes all three mechanisms and allow the parties to negotiate one, based on their preferences.

Transactions are categorized as low-value or mid-high value based on the amount of each transaction. Using public key cryptography penalizes operation costs when the real value of the transaction is low – the cost of public key operation exceeds the value of the payment rendering the transaction economically unfeasible.

Low-value transactions, also known in the literature as micro-payments, are payments involving small quantities; from a few cents to a Euro. These systems minimize the cost of transactions by using hash functions. One of the earliest such systems is PayWord [Rive96]. This is based on unidirectional properties of hash functions and implements a mechanism called hash-chains. Micro-payments are widely used in electronic commerce such as accessing web pages or pay-based online documents.

Mid-high value transactions are payments where the value of the transaction is high. Security is important since the falsification of even a single transaction represents a considerable loss. Such systems use mechanisms such as public key cryptography, replication etc to achieve desired levels of security and surety.

Security considerations: Currency management should provide mechanisms to resolve payment disputes. Two malicious behaviors that need to be addressed are: *authenticity* and *counterfeiting*. The former has two aspects: *Authentication* of the user and *integrity of messages,* that is, verification the messages have not been modified. *Counterfeiting* occurs when unauthorized entities issue and circulate currency. Public key cryptography is the most well known method to address these two aspects of security by encrypting data (against counterfeiting) and signing the encryption (authentication). [Okam92, MuNg01, Brand93] use Chaum's proposal [Chau90] to ensure authenticity and non-counterfeiting as well as maintaining anonymity of users. This increases the cost of each transaction.

3.2.4 Lessons learnt

Allocation is the process of distributing a finite amount of resources amongst a set of needful agents. Efficient allocation, both in terms of the solution quality (who gets which resources) and the computational efficiency, is a fundamental problem especially in large scale systems. A typical objective is to choose metrics that guide decisions and then develop procedures to determine allocations that are optimal with respect to the chosen metrics. Markets allocate resources to who value them the most. They also decentralize control, that is, participants make their own decisions and the system does not rely on centralized decision making. Allocation of resources is dictated by the price system and the control of computation resources is exchanged between applications according to market-derived prices. Consequently markets have received a lot of attention over the recent years and have been applied to the allocation of resources such as network, storage, processing time, and scheduling of computational tasks.

Systems such as [FLZ05] assume a *closed system* model where the set of users (or consumers) and the set of resources are tightly bound. They assume that the provider of a given type of resource is also a consumer of the same type of resource and propose a closed economy of Grid markets. This is an assumption that may not be made within Grid4All, for example, providers of CPU may want to consume storage or data.

Such systems also assume that resources are *divisible;* a compute node is time-shared amongst a number of competing applications. This is valid in a number of usage categories, but insufficient to provide a utility-model for VOs since issues such as isolation and security render sharing of compute nodes impractical.

Few systems address *co-allocation* or the guaranteed allocation of a bundle of resources. Most of the systems exhibit an architecture where each resource provider (in general assumed to be large clusters), auctions its own resources. Co-allocation implies simultaneous negotiations at multiple auctions (equivalently where a consumer engages in multiple bi-lateral negotiations with many providers) and may lead to an exposure problem where consumers have been able to allocate only a subset of the total bundle of required resources. Requests for bundles may also show complementarities; in this case, engaging in simultaneous negotiations increases the complexity for such consumers.

Systems are either completely *centralized* [CHUN04], providers and consumers trade at a centralized and long-living market place which matches offers and requests, or completely decentralized [FLZ05] [EYM03] [RV04]. The latter organizes bi-lateral negotiations between providers and consumers. This is perfectly acceptable in environments where there are a reasonably small number of *'large'* providers owning cluster

farms. Neither fit our requirements because centralized systems are neither tolerant to fault nor scalable and a completely decentralized system is not scalable.

Analysis of existing literature shows that market mechanisms need to be appropriately chosen to meet the demands of the targeted system requirements. [FLZ05] has chosen proportional sharing approach due to the system model where allocation of similar divisible resources (CPU), needs to be arbitrated amongst multiple price-anticipating users. The choice is based on criteria such as economic, computation performances and the ability to predict market behaviour. The choice of a mechanism and the rules governing the allocation is guided by the nature of resources, the number of different resource types and the demand function of the application. For example, consider a network routing model where the objective of the operator is to allocate resources such that the total cost of delay over all consumers is minimized. It is the interest of the operator to choose a mechanism that provides incentives to consumers to submit truthful preferences. This is because message sizes and resulting costs to the consumer when messages are delayed are information private to the consumer. Truthful extraction of this information helps computation of an optimal solution.

> ***Flexibility**: The first objective is to focus on a flexible framework that allows isolation of the market algorithms from the market place architecture. The architecture should provide support for design and development of different market mechanisms, support cohabitation of multiple market instances where the different instances may be governed by possibly different market rules and mechanisms.*

Sustainability of a Grid economy based on Internet markets depends on providing incentives to both consumers and providers. [KLAI05] doubts the usefulness of markets when real money is not involved arguing thus the failure of a number of approaches to transcend towards Internet settings. Effective currency management based on an open-loop economy gives incentives to providers to share their resources and gives incentives consumers to truthfully reveal their values.

Recent studies on the usefulness of market based resource allocation that does not use 'real' money, show degeneration to situations noticed within traditional systems based on priorities and resource shares, that of gaming the system [SNP05]. Closed loop environments [FLZ05] where a virtual currency is used as a mechanism to regulate bartering, is not appropriate for Grid4All. Closed-loop assumptions are valid when providers are also consumers and all users have (more or less) similar applications and usage patterns.

> ***Openness:** Grid4All needs an open economy based on a currency system backed by real money.*

In decentralized systems allocations are negotiated bi-laterally between consumers and providers. This has a disadvantage within Grid4All. The feeble quantity of resources owned by a given provider may in most cases not satisfy the needs of consumers. The other extreme is that of centralized and monolithic single markets. Such markets implement a specific market mechanism such as combinatorial or double auction mechanism. The logically centralized market place receives bids and offers from all the participants and employ algorithms to decide on optimal allocations. This solution is limited in the number of items and number of bids. Moreover it is impractical given the Grid4All context and environment where there may be no rigid and simplifying assumptions on when resources may be available.

Grid4All seeks a compromise between the two ends of this design space. This can be achieved through a market place that allows spontaneous creation and management of markets. There is also a second motivation. Formally a market describes the rules of participation, the rules by which prices are formed and rules that decide the allocation. The latter is referred to as the process of winner determination and can be assimilated to problems of optimization, in general complex. Recent years have seen the surge of research in this area; new formats and algorithms that allow matching buyer's preferences to that of seller's offerings have emerged. There is a large design space for allocation algorithms, and this dependent on the specific needs, in terms of traded resources, number of participants. Hence this project aims at developing necessary tools to support the emergence of structured electronic marketplaces for computational resources. Participants should be able to create markets (for example auctions) on demand, choosing the market mechanism and structure that suits their needs. The market place services should enable participants to

create markets on demand (here markets are intended as mediators such as auctioneers where participants may meet and trade).

> ***Dynamic**: Grid4All should provide tools and services to enable spontaneous creation and management of markets.*

3.2.5 Grid4All and Markets

The main objective of this task is to address the allocation of resources through markets. Market is a mechanism to coordinate and to match supply and demand. Within Grid4All, markets are used by owners of computational resources to sell and by consumers to buy resources such as processing time and storage. The organisation that we foresee is that interested parties (humans) create VOs that will execute specific applications and services to achieve the technical objectives of the creating parties. This has given the motivation for the work presented in Chapter 1 that describes a virtual organisation (in Grid4All context) as a first class and managed run-time entity. Section 3.3.1 presents some motivating use cases and scenarios that illustrate some objectives for creation of virtual organisations which are the consumers of raw computational resources (such as CPU, storage). VOs use these resources to execute applications that in their turn render services to end users.

The vision that we have for Grid4All is a system where not only resource providers and consumers but also 3rd party agents may spontaneously create *small* markets on demand. Small is characterized by the number of participants, the duration of a market, and the total volume of trade. The electronic resource market place should offer tools that enable the instantiation of markets and the discovery of these markets so as to be able to participate.

Successful operation of resource markets may also imply the involvement of other actors within the environment of the virtual organisation:

- Application developers: Our current belief is that applications need not be aware of the economic environment on which they execute. However it is expected that correct understanding of application performance and behaviour is essential for a correct usage of market based allocation. For example, video transcoding applications are easily parallelized by dividing the video file into video segments of appropriate lengths. Understanding of the behaviour of the application as more processors are added to it is essential to evaluate the correct quantity of CPU resources that should be allocated. Trade-offs may need to be done between the quantity bought and the price paid.
- VO administrators, initiators, and users: VOs acquire resources to execute its applications. Attribution of budgets to applications may imply the intervention of administrators and initiators. Users may also be required to contribute to the budgets of the virtual organisations. The budget and account management internal to VOs is an issue that we currently consider out of scope.
- Applications: The VO management handles applications as a managed element whose run-time requirements for resources may be directed through policies set by users and/or administrators. So we expect that allocation of resources through markets is transparent to applications.

The technical objective of this task is to provide services and software tools that enable the construction of open Grid resource market places. Consumers and providers of resources are the ultimate users of the market place and its services. Without prescribing specific behaviours for these roles, our objective is to develop proof-of-concept minimal automated buyer and seller agents that use the market place services to procure resources for the use scenarios that we plan to demonstrate within Grid4All.

This document presents the main requirements, state of art, and the architecture for open resource market places.

3.2.6 Organisation of the document

The section presents the organisation of the subsequent sections of this chapter. Section 3.3 presents the essentially non-functional issues and requirements that raise from the environment that Grid4All targets. It then presents a set of use scenarios and an analysis of the nature and properties of computational resources. Section 4 presents an analysis of market mechanisms, in particular auction-based mechanisms. It

then provides a classification of some important mechanisms and matches the targeted use scenarios to suitable market mechanisms. Section 5 presents the system architecture for the Grid4All market place emphasizing on requirements for an open platform. Section 6 presents a component based framework that for the design of the auctioneer (or mediator) role within a market place. This is driven by our assumptions that essential interactions between sellers and buyers of resource are mediated through an instance such as an auctioneer. Section 7 presents two major services: market information service that is an essential component for a large-scale system and a decentralized currency management service providing bank and accounting facilities. This chapter terminates with an analysis on the relationships with other main tasks of this work package that are essential to implement and deploy the market place services and a discussion on the future work.

3.3 Grid4All specific issues and requirements

The specific issues and requirements for a Grid4All market are derived from the project vision of a "democratic" Grid as a ubiquitous utility whereby domestic users, small organizations, and SMEs may decide to share resources by offering, selling or donating computing resources on the Internet. While some users may specialize in selling these resources, some others may offer and at the same time draw on resources, and some others may work without having to individually invest and manage computing and IT resources.

Dynamic virtual organizations may span multiple management domains, where each domain belongs to either a user member of the virtual organization and who has contributed its resources, or those of resources acquired from the resource market. Our goal is to support a large number of such simultaneous collaborative communities on the Internet; though the size of any one specific virtual community to be of a small or medium size.

Our objective is to enable on-demand delivery of resources to virtual organizations; computing in the Internet needs to be as ubiquitous as a utility. Implementation of resource market places in open environments such take into account major non-functional requirements described below.

Scale: considering a population in the scale of millions of users sharing their resources across the Internet either on a for-profit or a non-profit basis becomes a challenge in handling the complexity that derives from the quantity of resources, users and numbers of concurrent actions (for example requests for resources).

Heterogeneity: Open systems consist of a diverse range of resources (for example different processor architectures, speeds, middleware etc.), diverse range of applications and application demand functions (such as elastic demand for resources, real-time), diverse usage patterns, resource owner behaviors (for example may decide to trade resources at different times), and consumer endowments.

Dynamicity: Grid4All shall support the on-the-fly creation of highly dynamic virtual organizations where participants and resources can join and leave at will. The lifetime of a Grid service or collaboration might range from a few hours (e.g. an e-learning interactive session) to several years (e.g. twinned schools cooperating on an educational project).

The central issue in resource allocation is the nature of resources itself. The section 3.3.1 presents some motivating use scenarios to show the flow and interactions between the participants or actors in the trade for goods and later parts of this section give an overview and abstract properties of different types of resources. Later sections of this report show how the main requirements are being addressed:

- Heterogeneity: This is addressed through the use of ontology to describe resources and markets, through generic bidding specification, by providing an infrastructure that enables the cohabitation of different types of auctions. Furthermore we also address the interoperability issue raised by the presence of multiple auction (or market) types. Section 4 describes the generic bidding specification that has been designed within Grid4All. It also discusses the rationale behind the need to provide support for cohabitation of multiple market mechanisms (such as different types of auctions).

- Scale and dynamicity is addressed through spontaneous creation and dissolution of markets. This is accompanied by enabling services such as the market information service that allows for aggregation and scaleable dissemination of market related information. Section 5 describes the overall system architecture and in particular presents the enabling services to support open, scaleable, and dynamic market places.

3.3.1 Use case scenarios

The use case scenarios are useful to define the scope of market-based resource allocation in dynamic ad-hoc Grids; identify components and services that are essential for market-based resource management in different types of Grids (e.g. utility Grids, data Grids, computational Grids) and for different Grid applications, specify functionalities of those components and services and the interrelationships among them.

The main actors that are involved in these scenarios are: buyers representing virtual organizations, sellers representing resource owners, and also 3rd party actors such as market makers or market analysers. This section presents a market-oriented view of some use case scenarios, some of which are also described in some other tasks in WP2 (task scheduling in T2.2), and collaborative applications in WP4 and presents a market oriented view of these specific scenarios: sellers and buyers interested in acquiring or selling resources, market initiators and market designers.

Actors

An organized group of people constitute a *virtual organisation*: it can be defined as a rather stable collection of members such as users or administrators, policies, resources and applications or services that are executed within the context of the organisation. The virtual organisation can act as a *buyer* or a *seller*.

The group of people constituting a virtual organization use software applications to do their work that are more or less aware of the infrastructure; they can directly or indirectly interact with the Grid4All API and components. Applications require resources or services (goods in the market) on which they may execute and which they may use. Some goods might be acquired in the market and incorporated into the VO. Therefore the VO management system must sense the demand, evaluate the need, bid, and incorporate and later discard external goods. Therefore the VO management system is the ultimate *buyer agent*.

A person or organization may also decide to offer/sell goods (computational resources or services): it is represented by a *seller agent*. Goods have an owner and are traded (or in fact leased) by these owners at resource markets. Related to these mechanisms, there must be the role of market designers or engineers who are in charge of designing, implementing or adjusting new or existing market mechanisms. Furthermore, there might be other third party actors such as market makers, market analysers.

Generic scenario: Utility computing

Utility computing is a business model whereby computer resources are provided on an on-demand and pay-per-use basis. As the Utility Computing service is based on usage, computing resources are metered, and the user charged on that basis. Utility computing is sometimes also called *On Demand Computing*. Similarly applications executing within a VO consume the resources currently available to the VO.

Some examples are: An application is writing to a file until the currently available storage is nearly exhausted. As a result, additional storage is acquired and incorporated into the VO, transparently to the application that can continue writing (until the VO budget is exhausted). The same example could be applied to processing (machine capacity), where a node could be incorporated into the VO, or to services that could also be incorporated into the VO (for instance a service for the conversion for a new video format).

In this scenario, the interaction between the allocation request and the trading for additional resources could be more or less transparent: the affected component (the VOFS reported in D3.2 in the case of file storage) will request the VO management to incorporate additional goods (storage) into the VO, or in an more indirect model, a VO policy would define the threshold and mechanism for when and which amount of goods (storage) are acquired in the market.

Generic scenario: Batch computing

In the context of batch computing, a user agent submits a job specification to a job scheduler. The scheduler has to obtain the required resources (or otherwise fail the application). Some part of these resources may be available inside the VO, but for some other part the consumer agent may need to trade for additional resources and bind them to the VO for subsequent use by the scheduler (Task 2.4). This acquisition is mediated through the resource management component of the framework presented in Chapter 1.

The interaction between the resource manager (the allocation component) and the consumer agent might follow the same principles as in the utility computing scenario. After the binding of new resources to the VO, typically the scheduling service will then map application tasks to resources within the VO.

Application scenario: Collaborative Network Simulator

The Collaborative Network Simulator Environment (CNSE) application can be used to support the scenario "collaborative simulation of computer networks" described in section 4.1.3 of Deliverable 4.1. As stated there, the usage of this application within a grid context will allow overcoming several problems: first, the processing resources available will be arbitrarily large, and thus the complexity of the simulations will not be limited by this factor; further, storage will also be available on demand, and thus the number of simulations and visualizations can be that required by the educational objectives, and not restricted by locally available disks. Therefore this scenario can be seen as a combination of on-demand storage (group repository service) and task scheduling (simulation service).

Application scenario: task scheduling

Chapter 4 presents a task scheduling service where one typical example is to execute a video conversion task: a user wants to change format of a local video or perform some processing to the whole video (e.g. quality enhancement). Having a broadband connection but very limited local processing capacity, the application needs to divide the video in pieces and ship them to remote processors. For that, the scheduler needs first to know the resources available (which combinations are possible) and then obtain a certain combination of resources to proceed with the execution of the task. To obtain these resources, the buyer has to buy access to a set of computing resources. First it uses the market information service to know the status of available resources (volume, price, etc), then that information is passed to the scheduler who determines a few resource sets, decides, and then the buyer agent is instructed to purchase leases for the required resources. The purchase of these resources implies finding, joining and participating in markets, in multiple bids or in a bid requiring co-allocation.

Application scenario: supply of a video film

Another typical application scenario is the supply of a video film in a video-on-demand (VOD) service over the Internet where there are a set of subscribers that they can access through broadband connections. They choose a film and then the VOD service pipes the films to other users who want to watch the same video.

The buyer wants to buy access to a film. The buyer contacts its local market agent to start an auction. The buyer agent specifies the parameters – the type of auction that it requires, the schedule times, and the resource that it seeks – here the film. The buyer (the initiator) of the auction is given back a handle to later control the instantiated auction. The buyer also specifies the desired locations for the potential suppliers.

The local agents contact the market factory, and a market session for conducting the auction is created. The market session is regulated by two time-outs, a period for registration, and a period to receive bids. Once started the market session advertises itself at the semantic information service. It then waits for the suppliers to register. It registers only those suppliers who satisfy the constraints of the buyer (such as location of supplier). Once the time interval to collect bids has been reached, then it processes them and selects the winning supplier. In a sealed bid auction, no feedback info is sent to the participants via the MIS.

The market process prepares an *Agreement* and verifies that the winning seller and the initiating buyer accept the agreement through a message that is sent to both the winner and the initiating buyer.. The market session then terminates, and the agreement phase starts.

In the agreement phase, an Agreement Manager entity will handle all details required to make the agreement effective, including payment and providing access to use the resource. In this process, several actions are performed: resource access, where a reference to the good (a video film in this case) is obtained; billing; logging; payment: transfer of funds; and reporting via the MIS.

Video Supplier (bidder)

In this scenario, there is also the viewpoint of the (video) *supplier*. Supplier agents subscribe for market (for example auction) advertisements at the SIS. They can subscribe by a number of criteria. Interested supplier agents register and once authorized, they can participate. A supplier may (if needed) also download a Market Protocol from the Market factory the (auction) protocol that corresponds to the market process at which it has registered to participate. Each supplier then sends its sell bid to the market process and then waits either for a refusal or for the agreement.

Specific scenario: Buyer agent interested in acquiring resources

This scenario describes the typical course of action that a buyer agent will follow, but the description will be based on the task scheduling scenario.

The buyer agent wants to buy access to a set of computing resources to schedule a task such as a video conversion task. First it uses the *market information service* to query about the status of available resources (volume, price, etc), and that information is passed to the scheduler (application) who determines a few resource sets, decides, and then the buyer agent is instructed to purchase leases for the required resources.

The purchase of these resources implies:

- finding and selecting appropriate markets (involving a query to the *Semantic Information Service*), the most adequate market process may need to be instantiated from a *Market Factory* service,
- joining and participating in markets: registering and interacting with a market (a specific mediator process),
- Submit a bid expressed in a bidding language capable to describe a collection of resources demanded.

Specific scenario: Supplier agent offers resources

This scenario describes the typical course of action that a buyer agent will follow, but the description will be based on the video supply scenario. Supplier agents can sell their resources by expressing a sell bid (or offer). This sell bid can be sent to a specific mediator process by either joining a selected process [a] or created on-demand [b]).

Anyway, the supplier agent must first use the Market Information Service to query about the status of available resources (volume, price, etc), and that information will be used to construct a sell bid.

In case [a] the supplier agent would subscribe to marketplace advertisements (auctions) at the Semantic Information Service and wait for notifications of them. Interested supplier agents register at a Mediator Process (MP). Once authenticated and authorized, they can participate in the MP. A supplier may (if needed) also download from the Market factory the (auction) protocol that corresponds to that mediator process at which it has registered to participate.

Each supplier submits its sell bid to a market process by sending a message and then waits or query for that information. The information can arrive during the process (depending on the visibility rule of the auction mechanism, or at the end: containing a termination of the market or informing of an agreement. In case [b] the supplier agent will contact the marketplace to start (create) an auction.

Specific scenario: Intermediary creates market

This scenario describes the typical course of action that a third party agent (an intermediary) will follow to initiate a market. An intermediary can be simple (just act as a third party to initiate and drive a mediator

process) or complex (an intelligent market maker that also acts as buyer agent to resell (and aggregate) other resources).

It will have to choose the parameters for the market process, create and configure the market process, define type of resources to trade, and start the market process (this implies selecting the protocol). In case of a complex intermediary, it will also register as participant (seller).

Specific scenario: Market designer

This scenario describes the typical course of action followed to create a new market process. It essentially involves creating the market process description or template that should include the behaviour (protocol) of all parties (buyers, sellers, and auctioneers) to ensure the interactions work as expected. This template should be described and formally specified. Then the template should be registered and stored at the market factory and registered at the Semantic Information Service to be discovered. Any negotiation process in the marketplace will be configured based on one process template, determining the behaviour of each participant on that process.

3.3.2 Resources in Grid4All

It is clear that the resources that are traded within Grid4All are computational resources, that is, processing time, storage, and access to applications. The structure and choice of a market model is guided by the type and properties of the resources that are traded. Thus, it is important to understand the nature of the resources and their main properties. As an example, auctions for perishable resources such as for example flowers need to terminate quickly. The Dutch auction is typically chosen since its bidding rules favour early termination. Here the property of the resource helps sellers to choose the pertinent auction mechanism.

This section presents an analysis of the main properties of Grid Resources so as to understand the nature of the resources and identify those properties that may affect the choice of market mechanisms. Basing ourselves on analysis of literature we identify the correspondences between the resources and properties and the ideal allocation mechanism. Properties may be either intrinsic characteristics or a system choice. For example CPU is a resource that is divisible. If the system choice is to allow shared access (time-sharing mode) then a mechanism that allocates a percentage proportional to the bid is desirable.

Resource and request characterisation

Continuous or Discrete: Continuous resources are regarded as being (infinitely) divisible. Continuous or discrete is an inherent property of a resource. For example, time and energy are continuous resources given that they can be divided infinitely. Individual units of discrete resources are considered atomic units since they are not divisible. The CPU is a continuous resource in the sense that can be divided in infinite portions (percentage of CPU) though of course infinite divisibility is practically infeasible due to overheads. In fact, many systems trade CPU as a discrete resource, that is, CPU is traded in distinct time-slots where each time-slot is allocated to only one bidder.

Divisible or indivisible: Divisible goods are those resources that can be allocated partially. In the case that the bidder wants the entire good or nothing, then its bid is considered indivisible. This classification is symmetric to the continuous and discrete resources discussed above. However, divisible or indivisible resources are a matter of the resource allocation mechanism itself rather than a resource property. For example, network bandwidth is typically considered as divisible since it multiplexes the different flows at negotiated rates whereas storage units are traded as discrete units.

Single unit or Multiple unit: In a multi-unit setting the different instances of one type of resource are indistinguishable. In a single unit setting every item traded at the market is distinguishable from another. For example a provider may trade its 10 CPU units are indistinguishable units or decide to start a market that trades in 10 distinguishable CPUs (CPU1, CPU2 …). The choice of what the unit represents is a decision that the seller needs to make. It may decide to trade its processing time in units of two CPUs (here a buyer may buy only 2, 4, 6 … CPUs). Bidding specification needs to provide compact ways to represent the preferences of both buyers and sellers.

Single items or Multiple items: A single item refers to a resource that is traded as one atomic item. The granularity, that is, the definition of the 'item' is dependent on the market. For example, four CPUs may be traded in as an atomic item that may not be disaggregated. Grids also introduce the notion of *composite* resources (or items). Bidding specification should allow the expression of composite resources or services. For example, in practice it does not make sense to trade CPU and volatile memory as separate resources. Practically applications cannot use the one without the other and procuring them independently poses the exposure problem.

Sharable or not: Divisible resources may be either shared, that is, allocated to more than one consumer at a given time, or not. It is the choice of the market initiator to decide if resources are to be shared or not. Nevertheless as in the case of divisible resources, selection of an appropriate market mechanism needs to take into account this property.

Time factor: Grid resources are leased. Consumers may have constraints on when the resources are needed and the duration of allocation; similarly providers (especially within Grid4All) may trade their resources only for specific times. Hence the bid (and offer) must be able to specify the time ranges within which resources are required and their duration. A typical bid that needs to be supported is that of a consumer that requires two CPUs satisfying attribute description = {CPU>1 GHz, mem>1 GB, disk>20 GB} for 10 time slots between 10:00 and 18:00 assuming that time-slots are defined to be of 30 minutes. This needs combinatorial auctions. Within single item auctions, the time-slot may also be considered as the item.

Complementarities: The use of bundles allows bidding for combinations of resources. Consumer requirements are usually complementary, for example, a CPU and memory may not be disassociated. Complementarities are typically goods with super-additive valuations (v (A) +v (B) < v (AB)), since the sum of the valuations for the single resources is less than the valuation for the whole bundle.

Substitutes: A resource (or service) is a substitute resource for another insofar as the two kinds of goods may be consumed or used in place of another. Typically substitutes have sub-additive valuations (v (A) + v (B) > v (AB)). One good is a perfect substitute for another only if it can be used in exactly the same way, at exactly the same cost, and with exactly the same quality of outcome; that is, when there is no particular incentive for a customer to prefer one over the other. The notion of substitute is tightly related to the resource. For example, if an application requires one hour of CPU and is indifferent to the time of day when it needs the resource, then the time-slot may be considered as a substitute in a market that trades CPU resources in time-slots.

Linear Pricing or Bundle Pricing: Linear pricing sets the same price for each item of resource; bundle pricing sets the price for the entire bundle. Bundle pricing increases the complexity since allocation must choose those bundles that maximize the objective function. Bundled allocations need to be addressed by a combinatorial auction mechanism.

Resource properties and attributes

This section is based on the previous characterization and lists the main properties addressed in Grid4All.

Resources are considered as *discrete and non-shareable items* – resources are traded in *non-shareable units*. Continuous resources such as CPU are allocated in *discrete* and *atomic time-units*. A resource is in characterized in multiple dimensions – by *attributes specific to the resource* (such as the CPU clock speed) and also that of *time*. Resources may be traded at a specific wall-clock time – through auctioning of the time-slots; in this case, the definition of the traded item is characterized also by the *time attribute*. Some combinatorial auction mechanisms do not require this; constraints may be introduced in the optimization problem formulation to ensure that a given resource is indeed allocated to only one bid for a given time range. One issue is that of the length of the time-slot. It is clear that a fast CPU will be required for a lesser duration than a slow CPU; in the first specification we require that consumers specify the required duration.

Bidding refers to the preferences that buyers and sellers send to the markets. The bidding specification allowed by the market should allow a compact bidding for aggregation of resources and composition of

resources. Buyers (and sellers) may also want to acquire more than one type of resource, for example, an application may need CPU and storage resources. Bids that express more than one type are referred to as bundles.

Consumers (and also providers) may have constraints on the allocation; applications may be agile and adaptive to variable quantities of CPUs, however may have preferences (or even hard constraints) on the number of CPUs allocated and also where they are allocated from. Some parallel applications may only tolerate allocations that are multiples of 2; some applications may tolerate only shared memory multi-processors, in which case all CPUs need to be co-located on the same physical machine. Finally within Grid4All we aim at 'localizing' allocations when possible. Consumers should also be able to choose resources based on their locations. The bidding language should allow specification of such constraints.

Grid4All traded resources

The Grid4All marketplace is organized to trade raw computational resources and applications. At this stage of the project we address only the former, i.e., CPU and storage. Whether it be computational resource or applications, providers trade their resources as services. The section 4.3.4 provides the formal specification for the bids for resources such as CPU and storage. Specification of applications is a future work.

Bidding specification requirements

In single-item auctions only a single value, the price is communicated. Multi-unit, multi-item, and combinatorial auctions require generic preference structures. This section discusses the requirements and the design of bidding specification for Grid4All market place.

- Bundles: Typical applications require more than one type of resource (for example, CPU and storage), and also multiple quantities of each type of resource, bids should be able to convey preferences on bundles of resources.
- Substitutes: Typical Grid applications are malleable. Four CPUs of 3 GZ may be equivalently acceptable as eight CPUs of 1, 5 GZ (assuming that the application may terminate within its deadline in either of the two configurations). Two relevant class of languages are *OR* and *XOR* [Chev06]. OR bids allow consumers to allocate maximum valuation bundles over disjoint goods. An OR bid may submit multiple preferences; the valuation of the bid is the maximal sum of all contained disjoint bundles. This language can however not represent sub-additive valuations. XOR (exclusive or) bids also allow consumers to submit multiple and exclusive preferences (or bundles). XOR bids express the set of substitutes that a consumer is willing to accept.

[Chun04] describes a language that allows fine-grained resource allocation; bids may specify percentages of each type of required resource (40% CPU, 80% memory, 20% network, and 10% disk resources). A variant is to specify a *uniform share* of each resource in the bundle (40% share of all node resources: CPU, memory, network, disk). Consumers first discover resources by their configuration and bid for a specific node. Without precluding these scenarios, we favour bidding on absolute quantities rather than for shares. Sharing of compute nodes amongst multiple virtual organisations is complex in terms of security and management. An elegant way to address this is through the use of *virtual machines* instead of shares.

The bidding specification needs to capture participant's constraints (buying or selling) and their preferences – such as co-location of the allocated resources at the same provider, or in general restrictions on the minimum and maximum number of offers (from providers) that satisfy a bid (from a buyer). Proximity constraints can also be used to allocate resources from the same sub-network. Participants should also be able to specify that the time-ranges and time durations within which the resources are needed (so as to satisfy internal deadlines).

Two notions have been introduced in the bidding specification: composite and aggregate. An aggregate represents multiple units of the same type of resource. A composite represents bundling of different resource types. Aggregates may be conjunctive, the entire request quantity should be allocated or disjunctive -- it is sufficient to allocate a subset. For example, a compute node is a composite (CPU+Storage) and four Compute Nodes is described as a conjunctive or disjunctive aggregate.

A bid encapsulates the (i) description of the resource (i) quantities (ii) time ranges and duration (iv) the prices and the (v) constraints in allocation. The following code snippet shows how a resource is represented and figure 13 represents a traded item.

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace f = "http://axkit.org/NS/xsp/perform/v1"
namespace s = "http://www.ascc.net/xml/schematron"
namespace xsp = "http://apache.org/xsp/core/v1"

start = Resources

Resources = element Resources{ composite |aggregate
}
composite = element composite {
(element item {xsd:string},
element quantity {xsd:nonNegativeInteger})+
}
aggregate = element aggregate {
element item {xsd:string},
element quantity {xsd:nonNegativeInteger}
}
```

Figure 12 Aggregates or composites of resources

The current bidding specification in Grid4All has some limitations that hamper its reusability:

- Specific constraints are hard coded within the schema and there is no generic add new constraints,
- Pricing does not support complex schemas such as volume discounts,
- Time specification does not permit specification of constraints in the allocation of time slots; for example, a bidder cannot ask for CPU1 at *TSa* and CPU2 at *TSb*, where *TSb = TSa + 2*.

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace f = "http://axkit.org/NS/xsp/perform/v1"
namespace s = "http://www.ascc.net/xml/schematron"
namespace xsp = "http://apache.org/xsp/core/v1"

start = item
item =
  element item {
    element Id { xsd:nonNegativeInteger }
    | element nature { Storage | CPU }
  }
Storage =
  element Size {
    element Min {
      attribute value { xsd:decimal },
      attribute unit { xsd:string "MB" | xsd:string "GB" | xsd:string "TB" }
    },
    element Max {
      attribute value { xsd:decimal },
      attribute unit { xsd:string "MB" | xsd:string "GB" | xsd:string "TB" }
    }
  },
  element Throughput {
    element Min {
      attribute value { xsd:decimal },
      attribute units { xsd:string "Mbps" | xsd:string "Kbps" }
    },
    element Max {
      attribute value { xsd:decimal },
      attribute units { xsd:string "Mbps" | xsd:string "Kbps" }
    }
  }
CPU =
  element Memory {
    element Min {
      attribute value { xsd:decimal },
      attribute units { xsd:string "MB" | xsd:string "GB" }
    },
    element Max {
      attribute value { xsd:decimal },
      attribute units { xsd:string "MB" | xsd:string "GB" }
    }
  },
  element ClockSpeed {
    element Min {
      attribute value { xsd:decimal },
      attribute units { xsd:string "MHz" | xsd:string "GHz" }
    },
    element Max {
      attribute value { xsd:decimal },
      attribute units { xsd:string "MHz" | xsd:string "GHz" }
    }
  },
  element FLOPS {
    element Min {
      attribute value { xsd:decimal }
    },
    element Max {
      attribute value { xsd:decimal }
    }
  }
```

Figure 13 Item specification

Related Work

This section presents the type of bidding and market structure that is derived from the system model of some of the better known market-based resource management systems that have been reviewed in section 3.2.

	Tycoon	Mirage	Sharp	Grosu's	Catnets	Bellagio	Grid4All
Continuous or Discrete	Cont.	Discrete	Discrete	Discrete	Discrete	Discrete	Discrete
Single Unit or Multiple Unit	Single	Multiple	Single	Multiple	Multiple	Multiple	Multiple
Single Item or Multiple Item	Single	Single	Single	Single	Multiple	Multiple	Multiple
Divisible or Not	Divisible	Non-Divisible	Non-Divisible	Non-Divisible	Non-Divisible	Non-Divisible	Non-Divisible
Shareable or Not	Shareable	Non-Shareable	Non-Shareable	Non- Shareable	Non-Shareable	Non-Shareable	Non-Shareable
Time factor	Start Time+ Budget	Start time + Duration	Lease	Millions of instructions +Budget	Allocation time	Start time + Duration	Allocation time + Availability time
Complementarities	No	No	No	No	No	Yes	Yes
Substitutes	No	No	No	No	No	Yes	Yes
Type of Pricing	Linear	Bundle	Linear	Linear	Linear	Bundle	Linear Bundle

Table 1 Bidding capabilities of some market-based resource allocation systems

Tycoon allocated CPU resources proportional to the bids. The price of the resource is determined by the total bids placed and each participant is allocated a share of the resource in proportion to its bid. Here CPU is shared (divided) between the bidders. Most other systems allocate resources in discrete time units. Sharp implements a commodity market in which agents negotiate for a single unit of resource, for example, a given compute node at a given time and for a fixed duration. Many systems allow requests for multiple units of multiple items and also specify preferences of agents over alternative bundles. Mirage allows bids on multiple units of each requested resource.

Grid4All, Bellagio and Catnets allow bundle bids (bids on multiple items). Bundle bids are needed to allocate heterogeneous resources provided by possibly different suppliers. These need combinatorial auctions.

Time constraint is handled in one of the two ways according to the allocation mechanism: the unit allocated is a time-slot (of a resource such as CPU), that is, the market trades in time-slots of computational resources, or it is implemented as a constraint in the allocation in the case of combinatorial auctions.

In Tycoon bids specify a budget for the time interval when resources are required. Each bidder is allocated a percentage proportional to the total bids received for that time interval. Sharp allocates leases within the desired time range (the bid specifies the time range and duration that resources are required in that range). Bellagio, Mirage, Catnets and Grid4All include time in the bid specification. In Grid4All, sellers specify the time range within which their resources are available, and buyers specify the range within which they require the resources.

As far as we know, only Grid4All and Bellagio provide bidding support for substitutes and complementary resources. Consumers can specify the different bundles they prefer using XOR bids. Only Mirage, Bellagio and Grid4All provide bundle pricing whereas Tycoon, Mercatus, Sharp and Catnets request bidders to give per-unit of item prices.

3.4 Market mechanisms for Grid4All

This section focuses on market mechanisms and in particular discusses market models that match the requirements established in sections 3.2 and 3.3.

3.4.1 Background

Some definitions first; a mechanism defines a set of feasible strategies, which restrict the kinds of messages that agents can send to the mechanism and makes a commitment to use a particular allocation rule to select an outcome and a particular payment rule to determine agent payments, as a function of their strategies [Kal03]. Intuitively a mechanism design problem has two components: the algorithmic output specification and descriptions of what the participating agents require, given as *utility* functions over the set of possible outputs. A mechanism solves a given problem by assuring that some *acceptable* output occurs. In Grid4All we study auctions as mechanisms to solve resource allocation problems amongst suppliers and consumers.

There are two design goals in the application of mechanism design to auctions and markets. The first is *allocative efficiency* that seeks to maximize the total payoff across all agents. The second goal is *payoff maximization* that seeks to maximize the payoff to a particular agent (typically the seller). This is the optimal mechanism design problem.

The space of possible mechanisms is large. Mechanism also allow for multiple rounds of interaction between agents and the mechanism, and for arbitrarily complex allocation and payment rules. Given this, the problem of determining the *best* mechanism from the space of all possible mechanisms can appear impossibly difficult.

The *revelation principle* [Gib73, GJJ77, Mye81] allows an important simplification. The revelation principle states that it is sufficient to restrict attention to *incentive compatible direct-revelation* mechanisms. In a direct-revelation mechanism (DRM) each agent is simultaneously asked to report its type. In an incentive-compatible (IC) mechanism each agent finds it in their own best interest to report its type truthfully. The mechanism design problem reduces to defining functions that map types to outcomes, subject to constraints that ensure that the mechanism is incentive-compatible. However, the revelation principle ignores computation and communication complexity, and should not be taken as a statement that “only direct revelation mechanisms matter in practical mechanism design”.

The design of a computational economy involves tradeoffs between often conflicting design goals. For example, on one hand, we might wish to allow users to express very complex descriptions of resources they wish to acquire in the economy. On the other, computing efficient resource allocations over such resource descriptions could algorithmically be very expensive and hence untenable.

Markets furnish a solution to the non-functional requirements that have been set in section 4. The main system properties that are addressed through markets are:

- **Decentralization**: Auctions are decentralized in nature.
- **Distribution**: Processes can be distributed amongst participants.
- **Scalability**: Auctions require little global information.
- **Adaptability**: Mechanism can be adapted to allocate different items.
- **Efficiency**: Auctions have been proven to be efficient allocation mechanisms.
- **Fairness**: Auctions are perceived as fair mechanisms by users.

Our objective is to experiment with different market mechanisms, selected to satisfy the set of applications and use scenarios that we envisage for the proof-of-concept demonstration; first an expressiveness bidding language that allows agents to build complex bids is developed. Then, a set of allocations mechanisms that best match the requirements of applications are studied, designed, and validated across the target set of use scenarios. The choice of mechanism is based on the property that they satisfy (that are required by applications) and the trade-off on complexity. Typical properties are: satisfaction (full or partial), support for bundles and complementarities, support for expression of multiple preferences through XOR bids.

Auctions match bids (from buyers) to offers (from suppliers). Resources are *leased* to consumers for time durations where start and end times are specified by the. Bids and offers are sent to an auctioneer that determines the allocations; winning allocations satisfy the objective of the mediator process (revenue or surplus maximization). Bids and offers specify the time range within which the resource is requested (sold)

and the duration. Markets are inherently decentralized allocation mechanisms since each participant decides locally on how to bid, on what resources to bid, and which market to participate.

We focus mainly on auction mechanisms since the competitive process of auction mechanisms serves to aggregate the information about bidder's valuation and to derive the trade prices. The auction process consists of three main steps: bid reception, bid evaluation, and calculation of settlement prices. Iterative auctions conduct this process in rounds; each round refines the prices based on the current aggregate of supply and demand. The price feedback allows bidders to learn about the competition and reformulate their bids based on the current market prices. The literature on auctions identifies a wide variety of auction types, which we discuss below. One of the main issues in auction theory is the performance comparison of different auction formulations. Two main criteria when choosing an auction are revenue and efficiency – sellers typically choose an auction that maximizes the expected revenue. [WWW01] presents a complete characterization of auctions and proposes a classification schema for different auction mechanisms, based on different properties and capabilities of the mechanisms. The annex 3.10.1 summarizes this taxonomy the short discussion that follows is based on this taxonomy.

Discussion

The annex 1 has described different auction formats. The next step is to identify some of their capabilities. The English and Vickrey auctions are generally used to trade single items. Double auctions allow both sellers and buyers to submit bids at the same time; in general double auction trade multiple units of a single item. Combinatorial auctions allow bundled bids and guarantees complete satisfaction.

Incentive compatibility is desirable property that ensures that participants have an incentive to report truthfully their preferences and in general is determined by the pricing policy used. Pricing policies where the buyer's payment is independent of its valuations are incentive compatible.

The objective function determining the winning allocations is chosen based on the direction of the auction. Cost minimization is a common objective in (reverse) auctions to buy, such as procurement auctions, in which the auctioneer minimizes the cost of acquiring the items. Revenue maximization is a common objective in (forward) auctions to sell, in which the auctioneer maximizes the revenue obtained for the items. Surplus maximization, that is, the difference between payment collected and the amount paid to the sellers is typically an objective of exchange markets.

Continuous double auctions (or exchanges) are relevant to applications that require immediate allocation of resources as against clearing auctions that are more pertinent for applications that plan their resource requirements over time. The following table summarizes the auctions studied so far and the type of trading they permit. In the next section we evaluate Grid4All scenarios with respect to the mechanism characterization provided in table 2.

Auctions	**Type of Resources**		**Satisfaction**	**Clearing Period**	**Iterations**	**Structure**	**Econ.Properties**	**Objective**
	Items	Units						
English	-Single	-Single -Multiple	-Full	-Scheduled	Single round	One sided	IC (only if Single unit or non – uniform price)	-Revenue maximization -Cost minimization
Vickrey	-Single	-Single -Multiple	-Full	-Scheduled	Single round	One sided	IC (only if Single unit or non – uniform price)	-Revenue maximization -Cost minimization
Iterative	-Single -Multiple	-Single -Multiple	-Full -Partial	-Scheduled	Iterative	One sided	IC	-Revenue maximization -Cost minimization
Double	-Single -Multiple	-Single -Multiple	-Full -Partial	-Scheduled -Continuous	Single round	Two sided	-IC depending on the pricing policy. - No IC to both sellers and	-Revenue maximization -Cost minimization

Auctions	Type of Resources		Satisfaction	Clearing Period	Iterations	Structure	Econ.Properties	Objective
	Items	Units						
							buyers	
Combinatorial	-Single -Multiple	-Single -Multiple	-Full -Partial	-Scheduled -Continuous	Single round	Two sided	IC	-Revenue maximization -Cost minimization

Table 2 Auction characterization

3.4.2 Application scenarios

This section chooses three application scenarios from those earlier presented in chapter 4 in order to illustrate which of the above auction mechanisms (or combination) is the most appropriate and how the auction mechanisms can be used in each of the application scenarios.

Scenario 1: Image processing, an embarrassingly parallel application

Schedulers are a base for the Grid Computing. The chapter 4 presents the design of a scheduling service for a category of applications that are referred to as embarrassingly parallel – applications that may be decomposed into a set of independent tasks, each processing one part of the input data. Given a set of resources, the scheduling service provides an execution plan that minimizes the *makespan*. The targeted applications are photography batch processing and video encoding. The photography batch processing will use a recently developed algorithm for de-blurring photographs resulting from camera shake during exposure [Fer06]. The video encoding, the service will use the open-source MPEG-4 encoder *mencoder* [Mphq].

For our purposes, we refer to the three software modules – scheduling service, scheduled application (tasks of), and the Grid middleware that manages the application workflow as *application scheduler (AS)*. The AS requires the buyer agent to find suitable sets of CPU resources, given a maximum budget. The AS then evaluates the different CPU configurations (configurations are discriminated by attributes such as number of CPU, CPU speed, network closeness of CPU) to determine those that satisfy the user set deadline. The buyer agent acquires CPU resources conforming to the retained configuration from the resource market. These are then used to deploy and execute the target application.

The scenario requires a mechanism that provides allocations of multiple units of single resources. The application being elastic, it is not necessary that the employed mechanism guarantees full satisfaction (allocation of the total quantity of resources). A pricing mechanism that induces truthful bidding removes needs for strategic bidding by the application. The k-Double auction described in the section above turns out to be an adequate mechanism that matches this application. It is not computationally intensive as the combinatorial auction and may be implemented through simple sorting based algorithms.

Scenario 2: Network simulation, a computationally intensive application

Simulators are nowadays usually employed with educational purposes in many Computer Network courses. Use of simulators within the context of educational institutions is sometimes hampered by the following two problems. First, the computer resources available are sometimes not enough to carry out in a reasonable period of time the many simulations that can be launched simultaneously by the students. This limits the complexity of the simulations that can be carried out and, as a consequence, the possibility of illustrating many network scenarios that could be of interest for students. Besides, the storage resources available are sometimes scarce. However, the amount of data generated by some simulations may be very large. Again, this implies that this type of simulations cannot be carried in order to foster the reflections of students on many network scenarios. Here the requirement is for allocation of bundled resources to multiple students in a classroom.

Network simulators require bundles of CPU and storage resources for variable periods of time. A Combinatorial reverse auction with multiple suppliers allocate combinations of resources within required time slots to applications and provide complete satisfaction.

Scenario 3: A replica management service

In a virtual organization implementing a data storage and retrieval service, data object replication is a method to improve performance and data accessibility for the totality of clients on the network. Replica schemes need to determine how many replicas of each object are to be created, and to which nodes on the network they are to be assigned. Replica schemes affect performance of the distributed system and the choice of the replica site is of primal importance. Replica placement techniques should determine the network dynamics, specify objects that needs to be replicated, obtain access frequencies and patterns on the data object. Based on the above information the replica placement system determines replica allocation that maintains high data availability and accessibility for the users in the whole network.

The problem confronted is that of allocation of storage space for the replica. Market based controls of replica allocation is one family of methods that are being increasing used [Car02, Sam06]. In the auction setup each primary copy of a replica is a buyer. These agents perform auctions on need – as a response to replica placement needs and perform auctions to allocate appropriate storage. In this scenario, the traded item is storage. The relevant attributes are the quantity of available storage and the preferred network location of the storage.

This scenario motivates us towards an iterative auction implementing winner determination of one or more items amongst multiple objects for trade. There are many goods, but all goods are substitutes in some proportion -- one item (storage unit) may differ from another due to the capacity and also the network distance. Buyer bidding language requirements are mainly to express allocations (OR) of one or more storage units. Valuation of buyers will include the costs of transfer of data items to the storage and also the location of the storage unit in the network. Iteration allows the storage buyers to get a better understanding of the likely price for relevant sets of storage. An assumption that can be made is that of monotonic quantity changes – as prices rise, quantities cannot increase.

A summary of the chosen mechanism for each scenario is presented below:

- Scenario 1: Elastic consumers (VOs) desiring multiple units of CPU resource desiring an incentive compatible auction. Multiple virtual organisations seek to acquire resources in such a seller initiated auction
    - Given this scenario, its requirements are matched with the table 2. The k-double auction with pricing set to k at 0, is adapted.
- Scenario 2: Consumers (VOs) needing a complete allocation of a bundle of resources (CPU and Storage) required for variable number of time-slots
    - Combinatorial reverse auction with multiple suppliers, or combinatorial exchange auction (in case of opened to multiple buyers)
- Scenario 3: Seller desires to trade multiple units of storage. Storage units differ due to capacity and also network distance to consumers. Sellers seek to find an optimal allocation and aim to adaptively obtain information about bidder's preferences. Consumers have an elastic demand curve and full satisfaction is not a desired property
    - An appropriate multi-unit ascending auction needs to be selected.

3.4.3 Bidding specification

The Grid4All bidding specification is an XML encoded schema that represents bids that may be generated by buyers and sellers and submitted to the auction markets. A bid is expressed as a tree that consists of nodes that may be either: NonPrimitiveBids or PrimitiveBids. The root of the tree is a NonPrimitiveBid and each child may be either a NonPrimitiveBid or a PrimitiveBid.

PrimitiveBids are leaf nodes and represents an item (a grid resource) of bid. The leaf node includes

attributes to indicate the desired quantity of the item, the *leasing* attributes (start time, end time, and duration) and the price. The item may be either a simple resource, a composite, or an aggregated resource. Non-primitive nodes represent the relation between its children nodes, i.e., XOR, AND, OR. Figure 14 represents a bid structure, currently with definitions for storage and CPU resources.

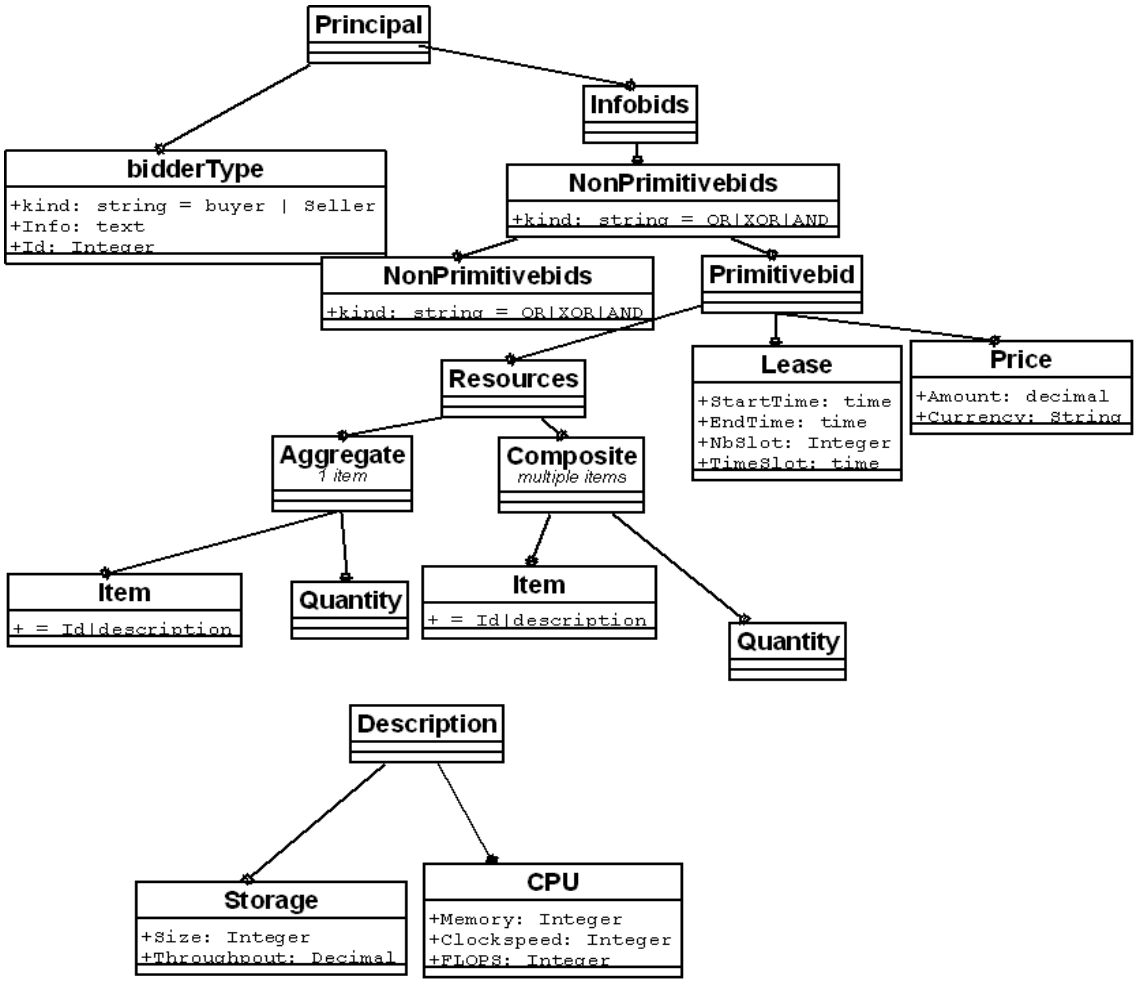
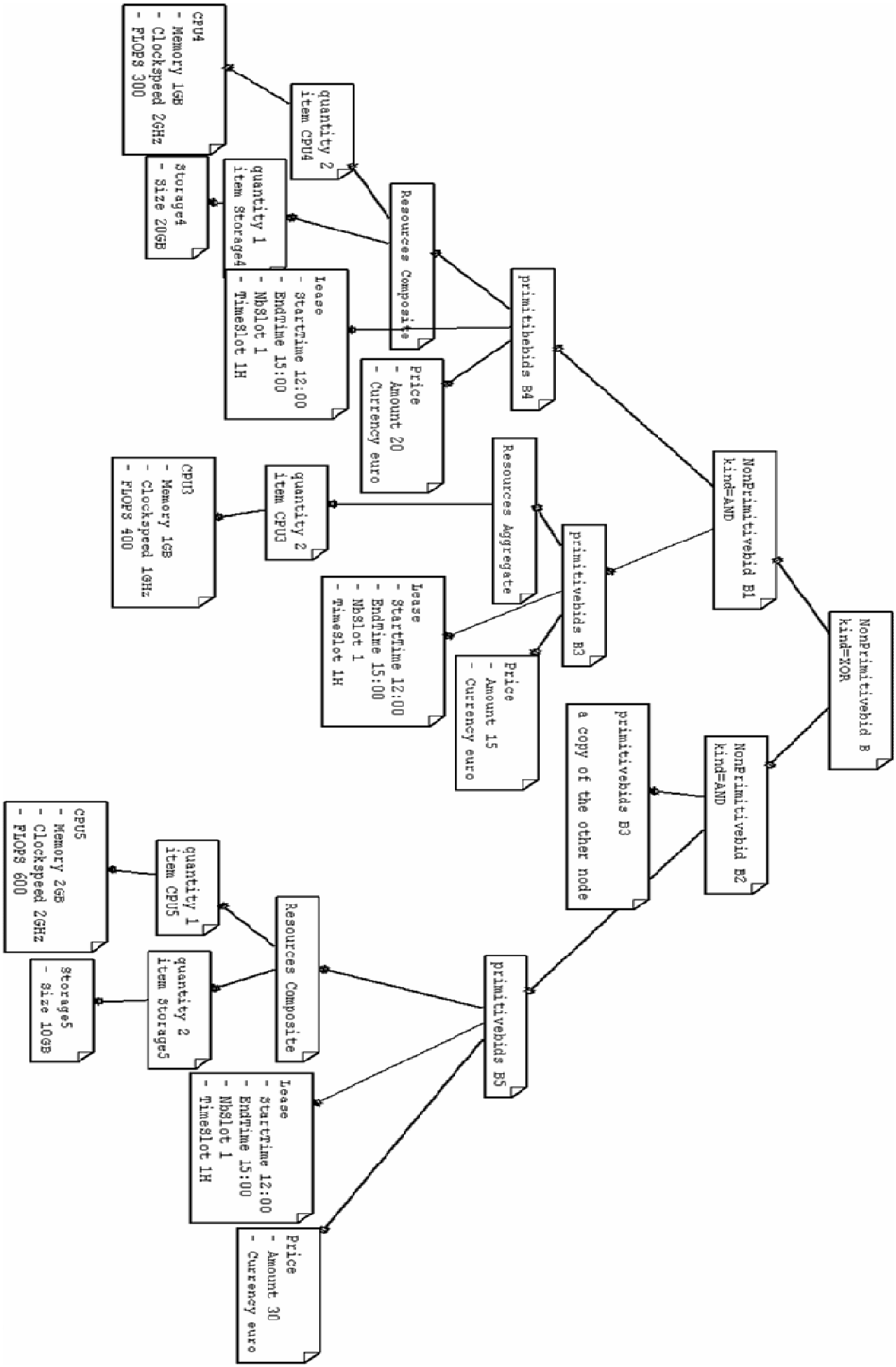


Figure 14 Bid structure

The example in figure 14 presents a bid for one of two configurations of resources to be used for a specified period of time (3 hours from 12:00 to 15:00). The bid expresses that the buyer requires one of the following: one CPU of 400 FLOPS, 2 CPUs of 300 FLOPS, or one CPU of 600 FLOPS. The following example illustrates the use of the bidding schema of a buyer's bid:

```
B =(B1 XOR B2)

B1=(B3 AND B4)

B2=(B3 AND B5)

B3 : 1CPU(1GB, 1GHz, 400FLOPS) from 12:00 to 15:00 for 1timeslot,
a timeslot=1hour. I'll pay 15euros.

B4 : 2CPU(1GB, 2GHz, 300FLOPS); Storage(20GB) from 12:00 to 15:00
for 1timeslot, a timeslot=1hour. I'll pay 20euros.

B5 : 1CPU(2GB, 2GHz, 600FLOPS); 2 Storage(10GB) from 12:00 to
15:00 for 1timeslot, a timeslot=1hour. I'll pay 30euros.
```


NonPrimitivebid B
kind=XOR
NonPrimitivebid B1
kind=AND
NonPrimitivebid B2
kind=AND
primitivebids B4
Resources Composite
quantity 2
item CPU4
CPU4
- Memory 1GB
- Clockspeed 2GHz
- FLOPS 300
quantity 1
item Storage4
Storage4
- Size 20GB
lease
- StartTime 12:00
- EndTime 15:00
- NbSlot 1
- TimeSlot 1H
Price
- Amount 20
- Currency euro
primitivebids B3
Resources Aggregate
quantity 2
item CPU3
CPU3
- Memory 1GB
- Clockspeed 1GHz
- FLOPS 400
lease
- StartTime 12:00
- EndTime 15:00
- NbSlot 1
- TimeSlot 1H
Price
- Amount 15
- Currency euro
primitivebids B3
a copy of the other node
primitivebids B5
Resources Composite
quantity 1
item CPU5
CPU5
- Memory 2GB
- Clockspeed 2GHz
- FLOPS 600
quantity 2
item storage5
Storage5
- Size 10GB
Lease
- StartTime 12:00
- EndTime 15:00
- NbSlot 1
- TimeSlot 1H
Price
- Amount 30
- Currency euro

3.4.4 Selection of market mechanisms

The choice of a target market mechanism is driven by the environment and usage for which the mechanism is intended – the selected mechanism must match the needs of the participants within that market. The selection criteria begin with the understanding of the requirements of the participants and the economic problems that the market should solve.

We address this by considering a set of application and use scenarios, identifying the type of resources, and the preferences on the resources that the applications may have, so as to retain a set of pertinent market mechanisms. The process we follow is summarized below:

Development Stages	*Description*
1-Analysis of the use scenario	✓ Identification of the desired properties.
2-Choice of suitable auction mechanisms based on	✓ Types of resources (multi-unit, multi-item) ✓ Need for immediate allocation or scheduled allocation ✓ Partial satisfaction acceptable ✓ Expected structure (how many sellers, how many buyers)?
3- Implementation	✓ Implemented in the framework described in section 3.6.
4-Experimental evaluation	✓ Iterative: Tuning of parameters and re-running of benchmarks
5- Understanding of allocation efficiency	✓ Analysis based on input data (bids)

Section 3.3 has presented a set of use scenarios that have been retained for experimentation within Grid4All. Section 3.4.2, has summarized the set of applications. This section selects an adequate auction design that responds to the requirements of these applications

- ✓ The single unit multiple item auction, where the auctioneer wants to sell a set M of resources maximizing the profit. The goods are distinguishable and each bidder may propose a bid on the subset of items (iterative auction on multiple objects)
- ✓ The multi unit single item double auction where the auctioneer wants to buy and sell a set M of resources and for each resource a number of indistinguishable units, maximizing the surplus. Each bidder posts a bid on the traded resource by specifying the desired quantity. The bidder provides a price one unit of the resource. (double auction on multiple units of a single resource)
- ✓ The combinatorial exchanges are particular auctions where the bidders can buy and sell resources/services. Each bidder posts a bid comprising of the set of items, the quantity for each item, and the price for the bundle of desired items. The price may be expressed either as a bundled price or as individual item prices. (double combinatorial auction)

Double auction design choice

The scenario 1 presenting an embarrassingly parallel multimedia application (Section 3.4.2) will use a k-Double Auction mechanism (k-DA) [WuWaWe98]. In this scenario multiple units of CPU need to be allocated, where each CPU may be requested for multiple time-slots. The traded items are in fact the time slots since each unit is bound to several time slots, i.e., a unit can be offered for multiple time slots. In a k-DA market resources are leased and the market is initialised by setting the minimum number of time-slots to be sold in the market (at least one) and the time ranges for which resources are to be traded. Bids specify the desired starting time, ending time, quantity of resources and the number of time slots of each resource. The application specifies the price of one unit of resource for one time-slot.

It is assumed that the market is configured to clear at a specified time and bids are accepted until this time. Participants may also withdraw their bids, but only before the clearing starts. The k-DA mechanism allocates (matches bids and offers) in granularity of a single unit. Hence complete satisfaction cannot be guaranteed in

the case where consumers require more than one unit of resource (or more than one time-slot). The only intrinsic guarantee is that at-most the maximum number of desired units are indeed allocated – resources may be partially allocated. We plan to extend this basic algorithm to impose complete satisfaction.

The introduction of multiple time-slots alters the notion of a 'single type of item': in fact an item (according to the basic k-DA) should be the CPU resource at a specified time-slot. Our intention is to extend this basic k-DA to allow participants to bid for resources over multiple time-slots, i.e., conduct the auction for not just one time-slot but a contiguous range. Since we extend to trade over multiple time slots, the basic k-DA needs to be extended.

An extended k-DA mechanism is presented below. The proposed mechanism is the first attempt to extend k-DA; it is rather straightforward and inefficient, as it may result in wasted allocation. For example, assume a buyer B1 who needs to allocate one a buyer B1 that needs to allocate one CPU for time-slots 1 and 2, and a second buyer B2 that needs to allocate one CPU for any two time-slots between the pre-defined start and end-time (start and end times of availability/requirement for CPU resources). Now seller S1 proposes one CPU that is available completely between the start and end time. Allocation of the first two time-slots of the CPU to buyer B2 results in wasted allocation. Based on experimentation on the first version that is described below, we plan to study and propose some heuristics that improves the efficiency.

The assumption is that there is one type of resource and the resource is traded in fixed time durations within a time range; for example, CPU resource traded between 12:00 and 18:00 in one hour units. For simplicity we consider that the time range is divided into a set of time-slots starting at 1, that is, the first time-slot starts at 12:00 and terminates at 13:00. The k-DA mechanism is divided in three phases:

Phase one: Generation of time-slot bundles

Bidders may request multiple units of the resource for multiple time-slots e.g., 1 CPU for any 2 time-slots within the first T time-slots. This is possible by the bidding specification described in the previous section. The first step of the mechanism (computed once at the initialization) consists of the generation of all combinations of time-slots across which the bid may be posted. If there are a total of *T* time-slots, the possible combinations are:

$$\sum_{i=1}^{i=T}\binom{T}{i}=\sum_{i=1}^{i=T}\frac{T!}{i!\times(T-i)!}$$

For example, if there are totally four time-slots numbered 1, 2, 3, and 4, and we allow bidders to bid for any combination, then potentially bidders may bid for (eliminating non-contiguous slots), implying that the market conducts auctions for each of these time-slot bundles.

1, 2, 3, 4, 12, 13, 14, 23, 24, 34, 123, 124, 134, 234, 1234

Phase two: Pre-Processing of composite bids

A generic bid is pre-processed to generate specific bids that can be posted across the possible time-slots. This step computes all the possible sub-bids that may be generated from the generic bid. The number of partial bids is: q * N where N is the number of requested time slots and q represents the number of requested units in the bid; a bid for 2 resources and 3 time slots gets pre-processed a set of sub-bids bid**XY** where the first index X represents the unit number and the second index Y represents the time-slot number of the bid (and not the time-slot number for the auction itself):

{{bid11} {bid12} {bid13} {bid21} {bid22} {bid23}}

Each sub-bid of a bid from a bidder is posted against the possible time-slot bundles generated at phase 1 according to the following rules:

1. A sub-bid is posted against a time-slot bundle of size 1 if:
    - If the time-slot number respects the time constraints of the bid

- The required initial time of the sub bid is less or equal than the initial time of the time range that represents the key.
- There is no other bid for the same unit number from the same bidder

2. A sub-bid is posted against a time-slot bundle of size greater than 1 if:
   - At least on of the time-slots in the bundle satisfies the time constraints of the bid
   - There is no other bid for the same unit number

An example of posting of sub-bids is presented below for a generic bid by a buyer that requires 2 resources for three time-slots each. It is assumed that the bidder required the time-slots between 12:00 and 16:00. The sub-bids generated previously are posted against appropriate time-slot bundles as follows:

Time-slot bundle	Value
1={12:00-13:00}	{bid11}{bid21}
2={13:00-14:00}	{bid12}{bid22}
…	
234={13:00-16:00}	{{bid11-bid12-bid13}} {{bid21-bid22-bid23}}
1234={12:00-16:00}	{}

Phase three: Clearing - Winner determination.

This phase clears the bids for each time-slot bundle and calculates the winners by using the K-DA algorithm: find the clearing price and then determine matching buyers and sellers. The clearing price is determined by aggregating the supply and demand at each price. The price at which the curves intersect is the clearing price. The bids from sellers below this price and the bids from buyers above this price are the winners. In fact a sorting algorithm is used to obtain the set of winning bids (and asks) [Bao03]. The winning sub-bids are removed from all the other time-slots at which they were posted in phase 2.

This phase progresses by computing winners for each time-slot combination; the important issue is the order of time-slot bundles to clear. A simple algorithm proceeds by numerical order of the time-slot bundles. This approach is neither economical efficient nor provides a fair allocation. Our objective is to study and propose heuristics for the clearing order.

The K-DA mechanism and its clearing algorithms will be implemented using the framework described in section 3.6.

```
HashTable cmb,Mbuy,MSell;
timeRange tr;
int tsize;
bid currentB,currentS;
List buy,sell,clear,win,lose,wintotal,losetotal,aux.
Int M,z,x,y;
String key;

tr=BidManager.getInitialOffer().getTimeRange();
tsize=BidManager.getInitialOffer().getTimeSlotSize();

//we get the list of buy and sell bids
buy=BidManager.getBuyBids();
sell= BidManager.getSellBids();

//computes the table that contain all combinations of time slots given a time range
cmb=BidManager.ComputeAllCombinations(tr,tsize);

//initialize the table with all bids/sell asks
while(!buy.isEmpty() && !sell.isEmpty()){
       currentB=buy.getFirstAndRemove();
       currentS=sell. getFirstAndRemove();

//given a bid(sell ask) computes all combinations for that bid(ask)
       MBuy=currentB. ComputeAllCombinations();
       MSell=currentS. ComputeAllCombinations();
```

Related work on double auctions

The literature survey presents several algorithms for winner determination in k-double auctions. [Bao03] provides a comparison of two implementations. The first algorithm makes use of four heap data structures. Heaps maintain the buy bids and the asks bids and offers operations to calculate the Mth and (M+1)st price. Clearing the auction is simply a matter of deconstructing the appropriate heaps. The second algorithm is based on the construction of an IPR-tree and maintaining two pointers (to the Mth and (M+1)st bid). The clearing process traverses the tree and inspects the bid at each node. If the bid should clear, the algorithm removes the node, an action that may require that the tree be re-balanced. In general this mechanism is adapted for single resource types; multiple time-slots imply that there are multiple items. In our case, we want to allow bids for contiguous time slots (sx,sy) within a time range (s1..sN), where $1 \le x \le y \le N$. In [Des04] a mechanism for pricing and clearing continuous double auctions in a peer to peer system is presented. The main feature is that consumers and providers broadcast bids for resources. Every buyer has incentive to trade with the announcer of the lowest sell ask that the buyer observed. Similarly, any seller would want to trade with the announcer of the highest observed bid. Trade is cleared at the middle price. Since peers do not have a global view of all the trades that occur in the system (when a trade is made between a buyer and a seller we cannot assume that they will communicate their price to the rest of the bidders), prices are updated when a peer observes a bid or ask from another peer.

[Bao03] present two efficient k-double auction implementations that allocate single resources for a single time slot. Grid4All approach allocates single resources for multiple time slots so that Grid4All k-DA can benefit from [Bao03] efficient implementations at the allocation of each time slot. The approach taken by [Des04] is fully-decentralized and does not facilitate the competition between providers that will more easily satisfy the needs of Grid4All consumers.

Combinatorial auction design

The majority of auction-based approaches examine scenarios in which typically only one type of resource is auctioned. Notable exceptions are [CHUN4] [SCH05] and [SCH06], which address allocations of bundles of resources. The former is also legitimate in that there is a vast range of applications that is typically dependent on only one resource type that may be storage, computational resource, video streams, or even a single Grid service, as is shown by the scenario 1 in the previous section. The scenario 2, describes a use scenario that requires multiple resource types -- computational and storage resource together in a bundle (all or nothing).

We leverage work from the combinatorial auction literature to allocate resources using a combinatorial exchange – an exchange is an auction where there are multiple sellers and multiple buyers who submit their bids. Within Grid4All, the majority of providers are expected to possess and hence trade small quantities of resources. Hence an exchange or combinatorial reverse auctions are more appropriate.

The requirements that the combinatorial auction model need to satisfy are:

- Suitable allocation of time-slots: Buyers should be able to express the time ranges within which they require the resources and to also specify the quantity of time slots that they need each of the resource. Unless other wise specified, the resources should be allocated contiguously in time (other wise applications may need to be check pointed and restarted).
- Consumer bids specify bundles: some applications may require that resources be co-allocated both in time and space. A partial allocation may be incompatible and unusable.
- Support for multiple attributes of resources: bids should be allowed to specify the quality and quantity of resources (such as the speed of the CPU)
- Multiple bundles in a consumer bid: A buyer should be able to express its preferences by submitting one or more bundles within the bid.
- Complete satisfaction for buyers: the system should allocate the entire bundle or nothing

This section presents a new winner determination model for a combinatorial auction that allocates contiguous time-slots, with the classical objective of maximizing the surplus -- this is defined as the difference between the aggregate of the buyer's valuations and that of the seller's reservation prices.

Mixed integer programming (MIP) techniques are powerful tools that allow handling many difficult decision problems. It is a linear programming technique used to perform optimization of a linear objective function under linear constraints. MIP is used for optimization problems in which only some of the unknown variables are required to be integers. These techniques are widely used to address problems in which certain resources have to be allocated optimally to certain customers, or problems in which facilities have to be located optimally so that all clients can be served in the best conditions. These tools are generic in the sense that they allow to model a wide range of problems and the resolution techniques can be adapted more or less easily to all problems. Although the models obtained are usually NP-hard, there exist many techniques to solve to optimality instances of reasonable size or to provide very good approximations.

In the Grid Market context, we plan to use mixed integer programming techniques to model the allocation of the resources to the consumers according to the bids. Several models will be derived in order to integrate in a progressive fashion the most complex features of the real situation. In the first models, we will first assess the difficulty of the problem by trying to capture the most important feature of the problem whereas the other aspects will be either ignored (relaxed) or approximated. In this document, we provide some elements of this first model.

A first MIP Model

If we assume, as a first approximation, that there is only one type of resource (CPU or storage for instance), a first model can be derived as follows: we denote by K the set of consumers and by L the set of sellers. Each seller $\ell \in L$ sends bids made of several resources $j = 1,\ldots,m^{\ell}$ proposed at minimum price $\overline{p}_j^{\ell}$ of one unit of resource per time-slot, a time interval $\overline{T}_j^{\ell}, \underline{T}_j^{\ell}$ the resource j is available. Each consumer

$k \in K$ bids a set of bundles consisting in a set of required resources $i = 1, \ldots, n^k$ with a maximum price $\underline{p}_i^k$ the consumer i is willing to pay. Each required resource is defined by a duration d_i^k, the earliest start time $\bar{\tau}_i^k$, the latest end time $\underline{\tau}_i^k$. We assume that each interval $\bar{T}_j^\ell, \underline{T}_j^\ell$ is divided into periods with equal length (time slot).

The problem consists in assigning some of the resources offered by the sellers to the consumer while maximizing the benefit of sellers. This problem can be modelled using the following variables:

• A binary variable $x_{ij}^{k\ell,p}$ equal to 1 if and only if the resource i required by the consumer k is satisfied by the resource j offered by the seller ℓ, and if this resource is in position p (corresponding to a period in interval $\bar{T}_j^\ell, \underline{T}_j^\ell$).

• Two continuous variables $\bar{t}_i^k$ and $\underline{t}_i^k$ defining the true starting and ending time of the resource i required by consumer k.

• A binary variable $y_j^{\ell,p}$ that indicates whether the resource j offered by seller ℓ in position p has been assigned to a consumer or not.

A first set of constraints is necessary to specify all relations between the time variables:

$$\bar{t}_i^k \geq x_{ij}^{k\ell,1} \bar{T}_j^\ell, \quad (1)$$

$$\underline{t}_i^k = \bar{t}^k + d_i^k, \quad (2)$$

$$\underline{t}_i^k - \underline{T}_j^\ell \leq M(1 - \sum_p x_{ij}^{k\ell,p}), \quad (3)$$

$$\underline{t}_i^k - \bar{t}_{i'}^{k'} \leq M(2 - x_{ij}^{k\ell,p} - x_{i'j}^{k'\ell,p+1}), (4)$$

$$\bar{t}_i^k \geq \bar{\tau}_i^k, \quad (5)$$

$$\underline{t}_i^k \leq \underline{\tau}_i^k, \quad (6)$$

$$p_j^l \geq \bar{p}_j^\ell, (7)$$

$$p_j^l \leq x_{ij}^{k\ell,p} \underline{p}_i^k, (8)$$

$$\sum_{k,i} x_{ij}^{k\ell,p} = y_j^{\ell,p}, \quad \ell \in L, \; j = 1, \ldots, m^\ell, (9)$$

Constraints (1) state that the starting time of the resource i of consumer k should start after the starting time of the resource j offered by seller ℓ if i is assigned to j. Constraints (2) state that the ending time of the resource i of consumer k is equel to its starting time plus its duration. Constraints (3) ensure that the ending time of resource i (from consumer k) should be less (earlier) than the ending time of the resource j (offered by seller j) as soon as i is assigned to j at a given position p (the parameter M is a large value that desactivates the contraints as soon as one of the x variables is not equal to 1). Constraints (4) ensure that: if both resources i and i' (from consumers k and k') are assigned to the same resource j (offered by seller ℓ), respectively in position p and $p+1$, then the ending time of the first should be less than the starting time from the second. Constraints (5) ensure that the resource i (of consumer k) start after the earliest starting time. Finally, constraints (6) ensure that the resource i (of consumer k) ends before the latest ending time.

We assume that all bids of the consumers should be satisfied and thus include an additional constraint:

$$\sum_{\ell,j,p} x_{ij}^{k\ell,p} = 1, \tag{10}$$

for any resource i required by consumer k. Finally, we have to specify some objective function that aims at minimizing the price paids by all consumers:

$$\min \sum_{\ell} \sum_{j} \sum_{p} p_j^{\ell} y_j^{\ell,p}. \tag{11}$$

Future work

Of course, this first model is very crude in the sense that it lies on a series of very strong asumptions. However, it is already a quite difficult problem which practical and theoretical complexity (probably NP-hard) should first be assessed. Then, some resolution method should be designed in order to solve some instances as efficiently as possible. After that, some extentions of the model will be considered, as for instances, cases with several types of resources, and cases where the objective includes some fairness criteria between the sellers and the consumers.

3.4.5 Conclusions

It is clear from the previous sections that the Grid4All market place needs to provide support for co-habitation of multiple mechanisms and allow participants to create markets as needed. Within the market place a number of *small* markets thrive simultaneously: small in the sense that the number of participants in any given market is much smaller than the total number of participants in Grid4All; the quantities of resources required may certainly exceed capacities of providers but are remain in the range of hundreds. Secondly there may be a large number of types of resources that are traded. Even though we have restricted discussion on computational and storage resources, our goal is to at a later stage define use scenarios including other resource objects such as video streams. In any case either type of resource is best allocated *close* to the consumer – when possible. Hence our approach to address scalability and heterogeneity is decentralization through these *small* and possibly *local* markets.

Decentralization is achieved by means of spontaneous market creation; i.e. participants should be able to create markets on demand, choosing the model, type and structure based on their own needs. Multiple small markets do not mean complete fragmentation or segmentation. We propose the implementation of a powerful market information service that gives participants awareness of the market activity and situation.

The second aspect of this approach addresses requirements to the market framework (presented in Section 7). Auctions are similar in their structure. The figure 6 indicates the flow of the bids within a simplified iterative auction process where the rules of a given auction mechanism governs the behaviour of each of the activities. Of course in the degenerate case, the number of iteration is 1. The differentiating rules apply to:

- ✓ Controlling the type of bids submitted – constrains of the bidding language allowed by the mechanism,
- ✓ The price updating rules,
- ✓ The termination conditions,
- ✓ The eligibility of the bids or the bid improvement rule,
- ✓ The feedback furnished to the participants (and also potentially the external world).

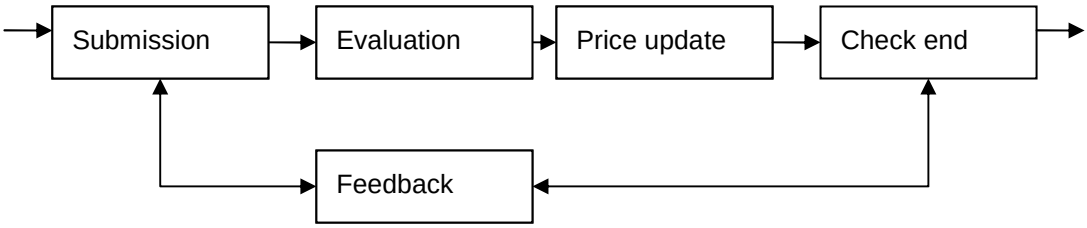


Figure 15 Auction process pipeline

The following aspects play an important role in the design of the market framework:

- The pre and post auction phase that consists of the initiation, the choice of auction rules, and the business transaction management functionality.
- The design of the auction allocation (or clearing) module, especially for the case of combinatorial auctions. A large number of variants exist – there is no universal support tool for the implementation of CAs.
- The bid construction and management plays a large role in the auction design. Some auctions may allow automatic generation of bids (similar to using of proxy agents), that pre-process and expand bids according to valid combinations permitted by the auction. Even though we currently focus only on price as an attribute determining the valuation, in the future it should be possible to integrate multi-attribute valuations, depending on the properties of the bundles.
- The process flow and control of the market/auction related activities: timing of bidding sequences, closing, clearing times are important parameters.
- System issues such as security.

The GRIMP (Grid4All Market Place) component based market framework governs the structure, control, communication, interactions of the components: (a) the framework should provide an easy-to-use and flexible interface to design and plug-in the rules of iterative different auction mechanisms. (b)The basic data structures and algorithms should be reusable in different environments and applications. (c) Allow for easy configuration and activity-control of the market process.

3.5 GRIMP System Architecture

This section presents an abstract model for the market place architecture. The purpose of this abstract model is to identify the generic components that are fundamental to implement market based resource allocations systems, ensuring that the architecture matches the specific requirements that arise from the addressed objective, that is, the allocation of computational resources to dynamic, ad-hoc virtual organisations. In the following section we recapitulate the design principles that the Grid4All market place middleware should satisfy so as to meet the requirements that have been discussed in earlier sections.

3.5.1 Design guidelines

To address the requirements discussed in the section 3.3, we have followed a set of design principles that allow us to meet them. An '*open'* market architecture is essential to address the scale and heterogeneity that is to be expected for grids on the Internet; the openness qualifies in the following aspects:

- Market mechanisms: no restriction to using a fixed set of market types,
- Market structures: markets may be forward, reverse, double,
- Market creation times: markets may be created when needed by any participant,
- Traded grid resources (and services): do not limit the set of resource types that may be traded.

Our first focus is on the architecture; architecture is concerned with the selection of architectural elements, their interaction, and the constraints on these elements and their interactions [PW92] and goes beyond the

algorithms and data structures of a specific computation, by focusing on the externally visible properties of software components [BCK98]. Abstraction aids in extracting the essential aspects of artefacts while hiding the irrelevant properties. Reuse is fostered by providing the essential framework that can be customized to produce several different artefact instances meeting different requirements.

We start by identifying the main components of the market system and the problems that they should solve. Then we abstract the design of the system as a generic architecture that may be customized to address a specific problem. We believe that 'Software Architecture' through decomposition of the system into components that perform elementary tasks and connectors that ensure that they interact is essential to address the requirements of Grid4All.

✓ Extensibility and adaptation

As discussed in section 3.3, the market place architecture should address different usage scenarios. Applying the principle of 'one shoe does not fit all' in the context of market mechanisms, it is clear that there is a need for separation of framework from the implementation of specific policies. Resource allocations and resource discovery are some components that need to strictly separate mechanism from policy.

✓ Standards when possible

An open environment implies adaptation to different platforms, a diversity of applications, and technologies. Usage of standards (when available) facilitates extensions and reuse; for example, usage of XML will be preferred to describe messages, configuration files, parameters to interface methods since this is flexible and language independent.

✓ Reusability and Encapsulation

In an open environment new market mechanisms and protocols will be designed or existing ones will be adapted to handle computational resources. This should not imply that designers rewrite from scratch their protocols.

✓ Layered approach -- from simple towards complex functionality

Traditional approach for resource allocation within Grid systems concentrated on furnishing a complex monolithic service – for example the implementation of inflexible protocols for co-allocation of resources. This approach implies that 'heavy' APIs and interfaces are furnished (system commands) that implement a specific behaviour. The system nevertheless is not tolerant to changes – or implies a major re-design. We take the approach to start from components providing basic services through basic interfaces. Complex behaviours may be achieved through the composition of simple behaviours [ZaP03]. This approach also facilitates the deployment of the services offered by the market place. Components may be deployed either locally or distributed amongst multiple nodes.

3.5.2 System Architecture

The Grid4All Market Place (GRIMP) architecture relies on the basic middleware services provided by WP1 to provide abstractions and hide the heterogeneity and distribution of the underlying fabric. GRIMP provides the market domain specific middleware to facilitate the development of Grid resource markets and market applications.

The main lesson that we have learnt from the existing market based Grid resource management systems is the need for adaptation and openness: (i) We should not restrict to complete decentralized bargaining based negotiations between interested parties (between a pair of provider and consumer) as has been the approach within [EY05], nor can we propose a system based on operating centralized markets – even if segmented over attributes such as resource types, users, locality etc. (ii) a great amount of progress has been made in market design and many forms of markets have been developed and experimented in the last years; the selection of a market design needs to be made based on the environment that it targets. These have given us the driving principle – that of a design incorporating open interfaces – to facilitate interoperability and flexible components – to facilitate development and deployment.

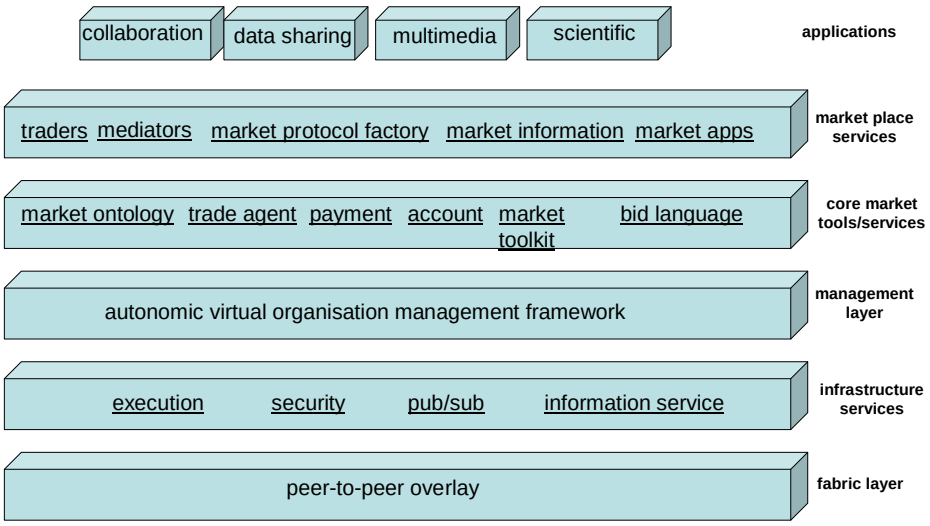


Figure 16 system architecture

This architecture shown in Figure 16 is composed of different layers:

- Application layer: this layer contains the end user applications such as virtual learning, simulation applications etc. Applications are hosted within virtual organisations – that is, the applications execute within the resources provided by the virtual organisations so as to satisfy the objectives of the VO.
- Market place services layer: this layer consists of implementations of specific market protocols and trading agents within the framework provided by the infrastructure layers. This layer also consists of market specific applications that may be developed using the functionalities provided by the infrastructure layer.
- Market infrastructure layer: this layer provides the basic functionalities to implement electronic market places. This layer also provides toolkits and frameworks to develop specific market mechanisms and algorithms.
- Infrastructure and services layer: this layer provides generic infrastructure required to implement large scale loosely-coupled distributed systems. This layer is not specific to GRIMP.
- Autonomic VO management layer: this layer provides the architecture based autonomic framework to construct self-managing and self-organizing virtual organisations.
- Fabric layer: this layer provides connectivity, communication, and low level discovery of the resources within the platform.

The open market place should provide the tools and services to create 'spontaneous' markets. Markets are initiated by the participants – providers, consumers (or also 3rd party mediators) on need. Such markets are typically short-lived and terminate once its objectives attained; Participants – much like within eBay [EBAY] should be allowed to create market sessions when needed.

Application layer

Consumers of the resource market are virtual organisations. These acquire resources that are required for the correct execution of its applications. Applications and services execute on the resources which are leased to the VO. The framework presented in detail in chapter 1 provides a certain number of architectural elements to implement management functions. Two important functions that are of relevance are:

- Installation, Deployment and Configuration of applications,

- Resource allocation.

Trading agents

Computational resources (processing time, storage) are acquired and sold by trading agents whose role is to negotiate. The negotiations are carried out between consumer agents on behalf of virtual organisations and seller agents representing providers that own resources. The essential actions of the consumer agent are resumed in the figure 17 and described below:

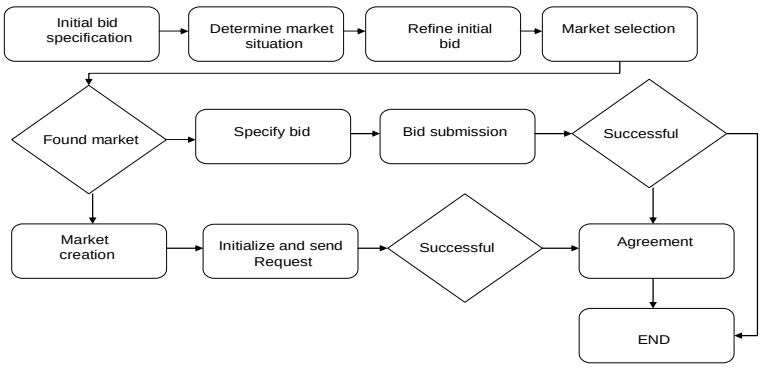


Figure 17 Main actions of the consumer trading agent

- **Initial bid generation:** This action is triggered in response to internal requirements for new resources. It consists of understanding the requirements of the application in terms of the type and quantity of resources, the deadlines, and the importance of the resources. As described in Chapter 1, the resource management provides a framework that implements a control loop based on sensors (monitoring the managed element), decision logic, and actuators. Monitors trigger events on detection of need for resources (such as CPU, storage). Decisions to allocate resources are based on this run-time information on the managed application. The consumer (or buyer) agent is invoked when resources need to be allocated from the market.The initial bid generation takes into account: (i) the current market situation (such as prices) obtained through queries to the market information service (ii) constraints on resources such as the quality of service attributes, the type of resources, their quantity, and (iii) policies internal to the VO such as those that decide on the budget. The generated *bid* is the service request that needs to be satisfied. To capture the information, we rely on the Grid and Market ontology that has been designed using the OWL [OWL] Ontology Language within this work package (Chapter 3).
- **Market selection:** Buyers (or sellers) select markets at which they may participate. They select markets that trade in resources that are compatible to their requirement and whose current prices are within their budget. Semantic representation (Chapter 3) of resources and bids are required to query and reason about the information during run-time.
- **Market creation:** This action is triggered if buyers do not find active markets that match. It consists of instantiating a new market session in a procedure that is similar to the well known *'request for proposals'.* The objective of this action is to create and instantiate a new reverse market.
- **Determination of market situation:** Agents needs to understand the factors that favour its trading opportunity. They may query the market information service to obtain information such as the current prices (or price curves) for the type of resources that it requires such that it can obtain information such as the initial price range. As an example, a VO that has a preference for locality of resources may choose to bid at markets situated outside of its preferred zones, if it understands that 'local' resources are in high demand.

- **Compilation of bids:** This action consists of compiling precise bids that may be submitted at a given market session. That is, the previous steps have identified markets where the consumer agent may participate. At this step, the initial bid is refined to comply with the constraints of the selected markets.
- **Bid submission and concessions:** The flow of this action is determined by the nature of the market at which the agent participates. For example, markets may be either iterative or non-iterative. In the latter case, bids are submitted once. In the former case, agents may need to make concessions in its bidding and evaluate an optimal strategy to revise their bids. At an incentive-compatible mechanism, this phase consists of determining the quantities based on the current prices. Consumers adjust their demands to current prices such that the utility is maximized. Within the generic framework we do not conclude on a single utility function – since this depends on the objective of the virtual organisations and the nature of the applications that execute within. In general (other than at incentive compatible markets) this action can be considered to be part of the strategy of the trading agent. Strategy consists of devising optimal moves in a negotiation and in general construction of strategy considers history of interactions, behaviour of participants, and understanding of market situation.
- **Agreement:** This final step is reached if the agent has succeeded. This step consists of accepting the agreement (Section 3.6), and complying with the requirements of the agreement protocol.

The dominant issues in the processes involving the consumer agent are (i) selection of an appropriate market session and (ii) bid evaluation. The three main interactions to be noted are:

- with markets: market learning (up) – partner selection, bid generation, market interaction (down),
- with applications: demand model of applications,
- with VO and its members: VO specific preferences, policies including that of pricing.

Market place layer

The market place is a forum for trade in Grid resources and services. The primary functions of the market place is to act as a matchmaker, to collect and publish statistics that trading agents decision process, encapsulate trade sessions and provide trusted third party services such as payment. The market place offers infrastructure, services, and tools to implement decentralized and independent trading opportunities between buyers and sellers.

Buyers and sellers need to interact with the market place to obtain the services that have been described in the previous section. The market place provides two main services: the *factory* that provides a *repository* of executable market protocols and the market information service. Initiating agents interact with the factory to instantiate trade or *market sessions* – much in the way that sellers interact with the eBay platform to create auctions for items that they desire to sell. Active market sessions are advertised at the *discovery* service where trading agents may discover them. Market sessions offer a generic interface that is independent of the market mechanism that they implement.

A common set of independent market infrastructure services facilitates the creation and growth of markets, by providing a common vocabulary, collecting and disseminating statistical information, by acting as a trusted intermediary, and by providing higher level services facilitating mediated interactions between consumer and supplier agents. Mediated interactions may have the following structures:

- Single supplier or forward auction markets normally initiated by a seller,
- Single consumer or reverse market normally initiated by a buyer,
- Double or exchange markets with multiple sellers and buyers that are initiated by a seller, buyer, or even an entity of neither of these two roles.

These patterns involve a *mediator* – referred also sometimes as the *Auctioneer*. The mediatory process implements the market mechanism, such as an English auction and offers a small set of interfaces to submit bids, query, and obtain feedback. A trading session can be regarded as an encapsulation of a negotiation process through which market services are delivered to participating agents. The market place provides a hub permitting the creation of market trading sessions. For example, a resource provider on a decision to sell multiple independent time-slots of a single CPU resource should be able to request the market factory to select suitable auction mechanism and create an instance of the market implementing the selected

mechanism. Then the provider should be able to configure the market, that is, set properties such as the type of resource that is traded (here CPU), the quantity, the initial reservation prices, the opening and closing times.

Protocol factory

In many systems such as [LAI05], [BAG01], the trading agent is designed and tailored to interact with one or at the most a few types of markets -- the agent is tailored for specific mechanisms, that is,, the buyer or seller agents is programmed to participate at a specific kind of auction. The objectives of the *market protocol factory* are two-fold: provide tools to developers to design market mechanisms and allow trading agents to access to executable and selectable descriptions of available market mechanisms. Recent years have seen a number of approaches – within or outside of Grid computing, that promulgates [BAR02], [RLGW06], [BBG06], unambiguous specifications of auction mechanisms in machine-interpretable forms. These are based on the foundational analysis on the structures of market negotiation mechanisms [SW03], [WWW01] that reveals the patterns inherent to *negotiation protocols.* Protocols are distinguished from *negotiation strategies* – the latter is the behaviour of the participant to achieve a desired outcome, where as the former defines the rules of encounter between the participants.

Figure 18 presents a conceptual architecture of the Grid4All market place; the market process is a mediating process implementing mechanisms (such as auctions). The process may be initiated by participants such as sellers or buyers. The market factory maintains a repository of executable market negotiation processes. We envisage the approach of a Web Services based market place – negotiation processes may be defined in [BPEL4WS] that can be executed through a process execution engine that enacts the market process.

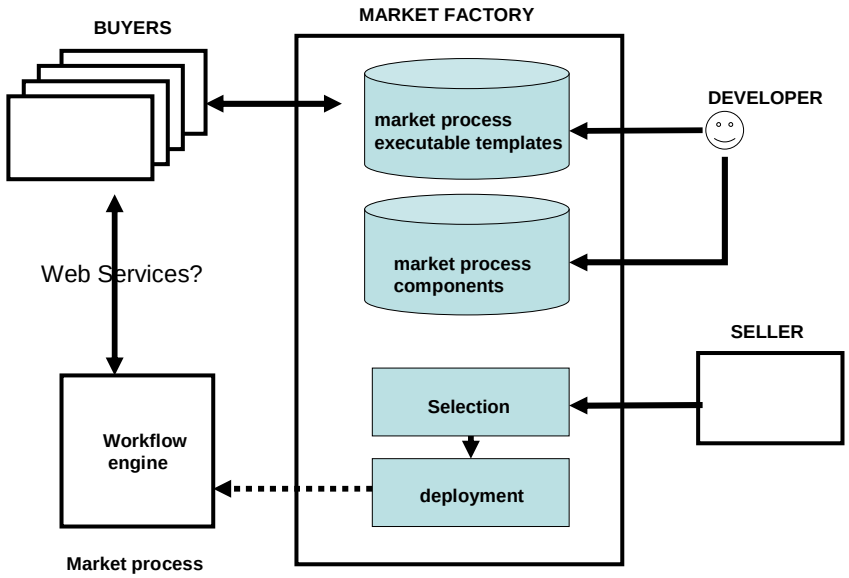


Figure 18 Conceptual architecture of market factory

Section 3.6 describes in greater detail the component based design of the market process and the issues related to the design of the market factory:

Market sessions

A market session represents an active instance of a market-based negotiation mediating between sellers and buyers. It is typically a short-lived process that is instantiated on need and terminates once the purpose

of creation has been reached (or there is complete failure). A market session may be *instantiated* by a seller (a forward market), a buyer (a reverse market), or also a 3rd party. In the last case, it is likely that the market implements an exchange between multiple sellers and multiple buyers. The *initiator* uses the services of the market *protocol factory* to select a specific market process.

To take part at a market requires that the participants have an understanding of its specific terms and structures -- the underlying market model is defined by the norms and rules that regulate its functioning [BAR02] – such as the type of bids that it accepts, the rules constraining bids, the visibility that participants have on the current bidding, and also the internal decision procedures that govern the pricing and allocation.

Section 3.6 discusses in detail the component architecture and framework for the development of different market models. It proposes a set of interfaces that address different concerns – configuration, parameterisation, and the different sets of interfaces to access the mediator process. Mediator processes need to offer interfaces that allow participants to register, to submit offers/bids, to query for current prices, and to be notified by the market as part of information feedback.

Once the mediator process has decided on the allocations and the transaction prices, the agreement phase is started. This phase establishes the *contracts* between each two ends of every finalized transaction. Agreements [WS-AG] are established through an *agreement manager* discussed in Section 3.6.

The behaviour of the protocol is determined by the sequence of invocations permitted by the market and the semantic content of the messages (or bids). Market mediator processes mainly execute the following activities: (i) receive bids (ii) evaluate bids and determine winners (iii) determine prices and (iv) and feed back information to participants. Mechanisms differ in the rules that govern these activities. The process may also iterate [NB06] in markets that determine optimal allocations by iteratively eliciting information from the participants.

Market Information Service

The Grid4All market place conceives a deployment scenario where there are multiple running 'independent' markets – each individual market by preference and construction will have a small number of participants who engage in trade. In this scenario a key aspect is that of dissemination of market related information. Agents will rely on this information to aid in their internal strategies – where to create a market, what profile of resources to request, what time to create markets, what guiding prices etc. This requires a performing market information service that is able to provide statistical and aggregated information to the participants.

Motivating examples of statistical information are:

- Number of suppliers for a given service/resource type
- Level of activity within the market indicated by volume of trade of a given type of resource, service
- For a given service/resource type, the going prices(min/max) on the market by location/zone/proximity

Section 3.6 presents the component interfaces of the market information service.

Payment and currency

Grid4All consumers may make payments to a provider in a currency accepted by the provider. The payment service offers a unified interface to payment, isolating the rest of the components from the particularities of the payment mechanism. It also, generates the required logging/auditing information.

The payment will be made through a Bank service which issues currency and maintains accounts for all traders who have subscribed (see above). Currency may only be issued by this trusted Bank service. Currency may only be transferred once – as payment from a consumer to a provider through an account maintained at the Bank service. Each registered/subscribed provider and consumer will have an account at the bank.

Section 3.6 presents the detailed architecture and the interfaces of the payment and bank service;

Bid specification

A bid expresses the preferences of the traders in terms of quantity, quality, and the prices. The bidding specification that we have designed takes in to account the following requirements which are necessary to request and offer Grid resources:

- Specification of 'bundles' of goods: Typical applications require a combination of resources, where the valuation may present complementarities. For example, an application may require both storage and CPU and not independently one or the other.
- Possibility to specify multiple bundles or configurations: Typical applications may accept one amongst multiple resource configurations. For example, a given application may adapt itself to (a) A CPU of 1gz and 2 giga of RAM memory or (b) A CPU of 2gz and 1giga of volatile memory.
- Support for multiple attributes. Some relevant attributes are:
    - Specify time ranges within which resources are available and required: Computing resources are reusable resources – reusable over time. Resources are acquired as leases covering specific intervals of time. Applications may be constrained by their internal deadlines on when resources are to be allocated – resources allocated too early or too later may not be of use
    - Specify the quality of the required resource: Resources are heterogeneous in terms of the amount of work rendered, or the throughput attained. CPU resources range from those that process billions of instructions per second towards that process a few hundred millions per second.
    - Specify the location of the required resource:
- Constraints on allocation
    - A minimum and maximum quantity: Applications may require multiple units of a given resource and hence would need to specify the desired quantity. This may also be expressed using XOR bids (multiple exclusive preferences), but nevertheless this constraint give a compact representation to the bid.
    - Minimum and maximum level of aggregation: Applications may require resources by specifying the aggregated capacity. For example 1000giga of persistent storage. Nevertheless the performance of the application may be related to the total number of independent units from which the storage is allocated.
- Specification of per-unit or bundle prices

3.6 Market framework

The Section 3.5 has introduced the different functions and services that need to be provided by the Grid4All market place platform. One of the functionalities can be compared to that of a traditional auction site or market portal that allows creation and instantiation of mediated trade sessions. This section describes the component architecture and framework of mediated markets. A minimum framework should provide the basic functions and components, that is, market specific such as bid management, clearing mechanisms, system specific such as market deployment, participant registration, and economic platform specific services such as agreement contract management.

3.6.1 Approach

We proceed by separating the market design from market mechanisms design. The market framework design includes the economic and system processes; the former includes the market mechanism that encodes the market specific rules and protocols. The objective of the framework is reusability, specialization, and rapid prototyping of new market designs.

Our approach follows a component-based design. Market trading mechanisms can be complex. Their complexity can be controlled by combining several specialized software components. A component-based approach promotes distributed and autonomous development as well as flexibility and the reuse of the developed software components for designing new marketplaces.

Each component has a specified role within the market framework, that is, corresponds to different functionality, either market specific (for example a given clearing algorithm or pricing) or that of a generic

infrastructure service (for example registration and authentication). Components encapsulate distinct aspects of the market so they can be customized and replaced independently. The integral marketplace architecture can be obtained by assembling the different components and binding their interfaces.

Participants interested acquiring or selling resources need to search for a suitable market where they can buy/sell, by properties such as type of traded resources, price, quantities. An open and flexible marketplace should not require that participating agents are hard coded to negotiate and interact with all possible market structures and mechanisms. They will need some support to adapt to a given market that executes a specific type of market negotiation mechanism. When trading agents search for suitable marketplaces the only restriction is to commit to some shared and common knowledge that gives an explicit representation of negotiation protocols. A neat solution for this is to have a shared factory containing executable negotiation protocols that may be selected and then instantiated.

This representation must allow participants wishing to get involved in a negotiation process to be aware of the interaction protocol and the negotiation rules governing the messages exchange through the specified APIs. The market protocol factory should be designed to meet the following needs.

- Formally specified market negotiation protocols should be made available in a machine-readable format at the repository.
- The executable negotiation process should allow protocol specific configuration,
- The market initiators should be able to choose from available market protocols,
- Participants at markets must be able to upload the process (protocol) corresponding to their roles so as to be able to participate at a given market implementing a given protocol or mechanism.

Requirements

Participants wishing to trade on grid resources interact according to a *selected* market negotiation mechanism. They may negotiate either through direct interactions or indirectly through a mediating process (referred to as an auctioneer process). Actors in this process are providers (supplier agents), consumers (buyer agents) of grid resources, or third party participants (brokers/mediating auctioneers, intelligent market makers, etc.). They may either participate in running market sessions or may create new market sessions.

According to these requirements, a Grid4All marketplace framework should provide the main following elements (see figure 10):

- ✓ **The Semantic Information System** (SIS) described in detail in Chapter 3, enables participants to discover suitable markets where they can trade the desired grid resources. The SIS uses ontology to model the traded grid resources (computational resources, storage resources and services), the exchanged messages between the market participants (bids, offers, orders) and the market itself.
- ✓ **The mediator process**, that we will henceforth simply refer to as *market*, provides some common specific elements of the market processes (economic and system tasks) in order to achieve design and implementation of different market behaviours. It enables handling and management of offers and bids from the participants, as well as considering the different (economic) steps of the used trading mechanism. Offers and bids are specified using bidding language based on an XML based schema.
- ✓ **The Market Process Factory** (MPF) is a repository that manages templates of market processes – these implement different types of mechanisms. The objective is to allow markets initiators to choose an adequate protocol to execute and allow the other participants pick the correct templates that they may use to interact with the instantiated market. That is they have to upload the required templates from the local MPF. Logically each market negotiation protocol can be looked upon to be formed of a set of templates corresponding to each of the permitted roles within that protocol.

Based on these assessments, our objectives are double: first, we have to focus on the market architecture, and then the factory of protocols templates has to be specified.

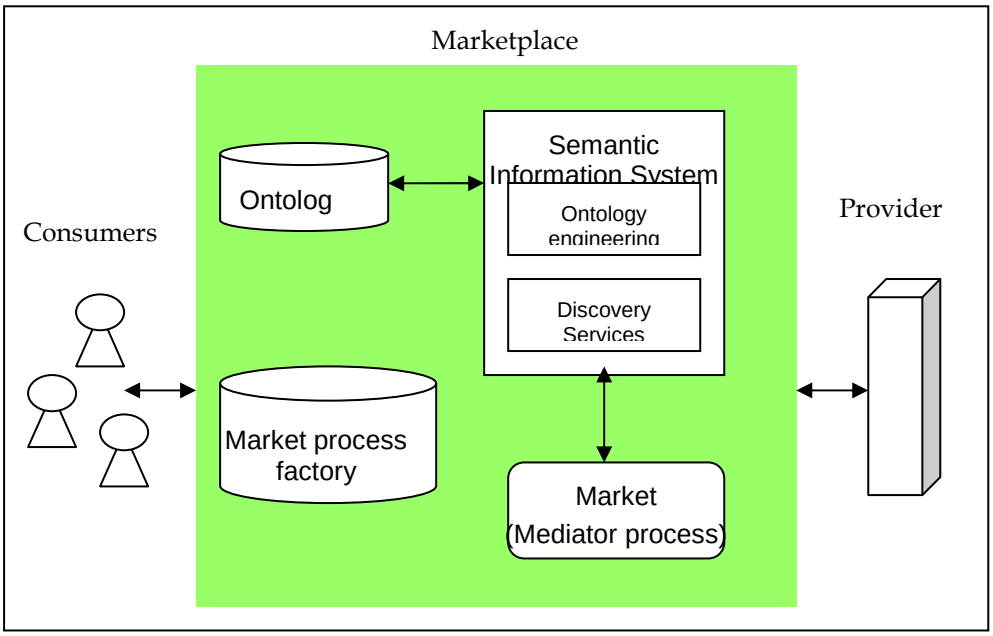


Figure 19 Grid market place architecture

3.6.2 Component based market framework

In this section, we first present the main functional components of the Grid4All marketplace framework. We give after a detailed presentation of the Fractal component model that will be used to build the described components.

Basic market components

An abstract view of mediated markets (inspired from the GEM system [Reich 98]) distinguishes mainly three basic layers: the system layer, the market layer and the participants' layer.

The system layer is the lowest layer in marketplace framework. It provides general infrastructure services to the upper layers, such as security services for authentication of participants, market recovery mechanisms, persistence services, and an execution environment.

The market layer deals with "economic" concerns and includes components for basic trading mechanisms:

- Registration and admission control: This layer has two roles, authentication and authorizing participants. The market accepts incoming orders only if they belong to an authorized participant (for example registered). Registration can also be constrained through rules limiting number of participants.
- Validation of offers: Incoming orders are filtered according to bidding rules (exp. In an English auction, a new bid must be greater than the previous bid) to accept only those that are conformant to the bidding and activity rules of that market. Accepted bids are organized in a storing structure in order to be handled by the market to determine winners.
- Determination of winners and prices: The market has to match bids with asks according to its clearance mechanisms. This component matches bids and orders and determines the prices that will

be paid by the winners. The latter is determined by the pricing policy (best price, second price etc.) that is active. Once all orders have been handled, the market price quotes are updated.

- Agreement and information feedback: An information component controls information that has to be available to traders at different stages of an auction and notifies the participants in the transaction about their order status. At the end of the transaction, the winner is notified and agreement procedure is triggered.

The Fractal model [Objectweb] has been used to design the mediator framework. It is a modular and extensible component model that can be used with various programming languages to design, implement, deploy and reconfigure systems and applications. The goal of Fractal is to reduce the development, deployment and maintenance costs of software systems. The Fractal model provides support for hierarchical components (components may be nested), is reflexive and provides full introspection and intercession capabilities, and is open, that is, extra-functional services may be built through the notion of the control membrane. Each Fractal component specifies:

- The interfaces that allow access to the components. Each component has a set of interfaces that can be provided or required by the component. The interfaces define and express the dependencies between the components in terms of required/client and provided/server interfaces. Each Fractal interface is associated with one component and has a name and a signature. An interface can be mandatory or not, simple or multiple (collection).
- Connections or bindings between the components that control the data flow between them. The use of an architecture description language (ADL) allows the assembling of the market components according to their specified interfaces. The resulting assembly represents the software architecture of the market process. The market framework architecture will be designed using Fractal ADL.

3.6.3 Market process

For the purposes of this section, we define the *market* process as the mediating process that implements a market mechanism (such as an auction), and based on the received bids from its participants determines the final outcome of trade between the buyers and sellers. Markets may be of different structures depending on the set of participants; forward markets with one seller and multiple buyers, reverse markets with one buyer and multiple sellers, and exchange markets with multiple buyers and multiple sellers.

The objective of a software framework based on modular components is that it facilitates specialization and can be used to implement a wide variety of market negotiation mechanisms. [BAR02] has presented a framework for the description of generic negotiation processes that capture the common aspects of a variety of types of market based negotiation. They derive that all negotiation processes involve:

- The object of negotiation,
- A set of rules that fully specify the negotiation mechanism and capture the wide variety of mechanisms in a structured way. [WWW01] and [SW03] have compared and characterised a wide variety of auction and negotiation mechanisms, paving the way for a structured approach to the design of market negotiation systems,
- The interaction protocol that defines the permissible flow of messages between the participants.

To illustrate the execution of negotiation processes two types of auction mechanisms are represented through activity diagrams. Figures 20 and 21 show the negotiation process of two types of auction mechanisms: a single bid auction and a continuous double auction. The auction process is called the mediator, since it receives the bids from buyers and sellers, and applies rules to determine the winners. These diagrams show the process executed by the traders and the mediator. These diagrams assume that the negotiation initialization has terminated and participants have registered.

Once negotiation has been initiated, the mediator process can accept bids; the auction process executes a set of activities that are triggered by events such as incoming bids or scheduled time-outs. The conduct of each activity is regulated by the rules pertaining to that activity. Figure 20 shows a single-shot auction where the market collects all bids from the registered participants; bid submission rules constrain the preferences proposed within the bid. Clearing (or the module determining the agreements), is scheduled by the rules

such as, at a fixed time interval from the opening or when all registered participants have sent their bids. The clearing module determines the winning bids and pricing rules govern the transaction prices. The results are announced to the participants, this again condition to the rules of the market (visibility and display rules). The continuous double auction shown in figure 21, allows participants to send bids at any time – subject to closing times of the market. Bids are cleared immediately on reception.

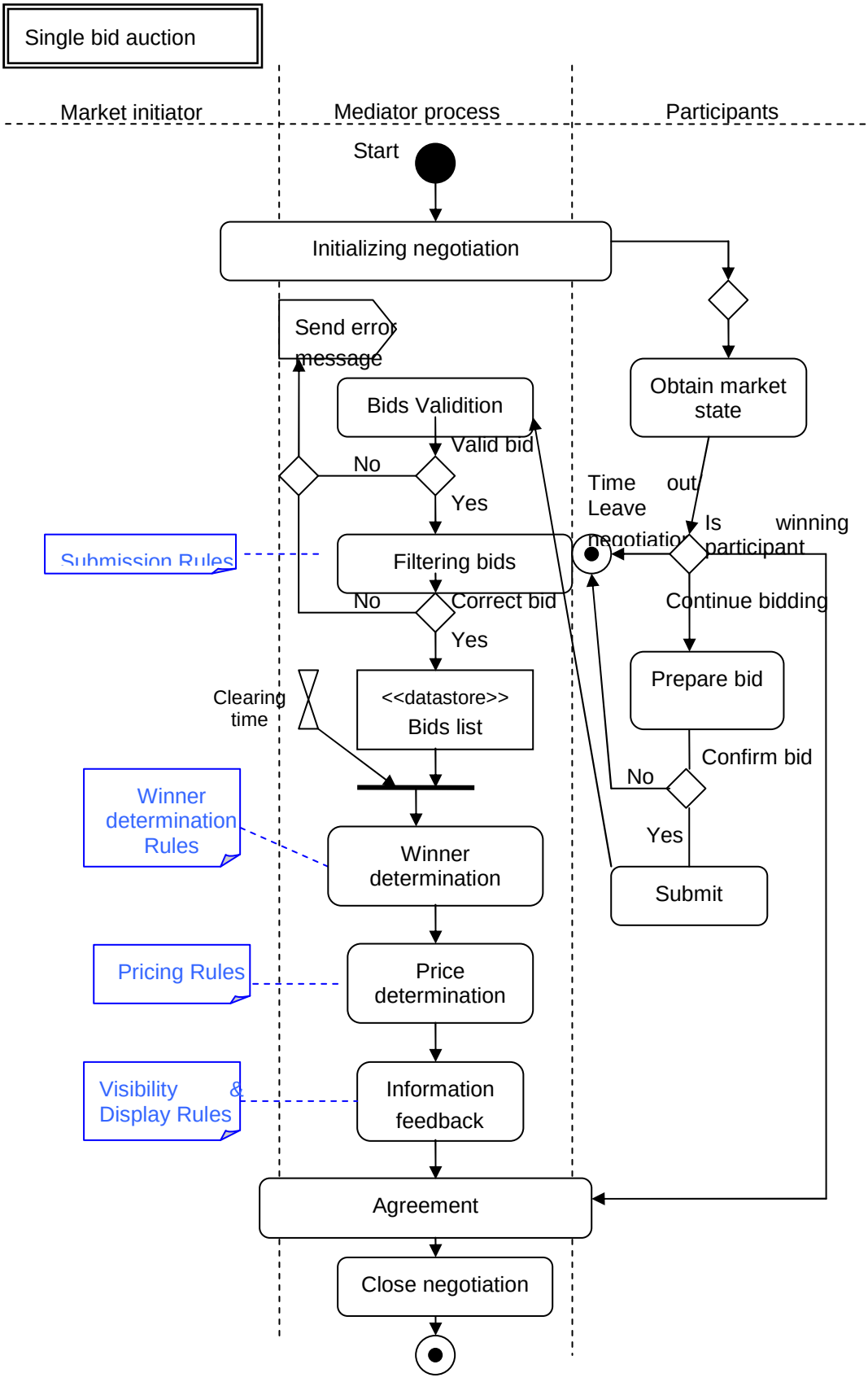


Figure 20 Activity diagram of a single-bid auction

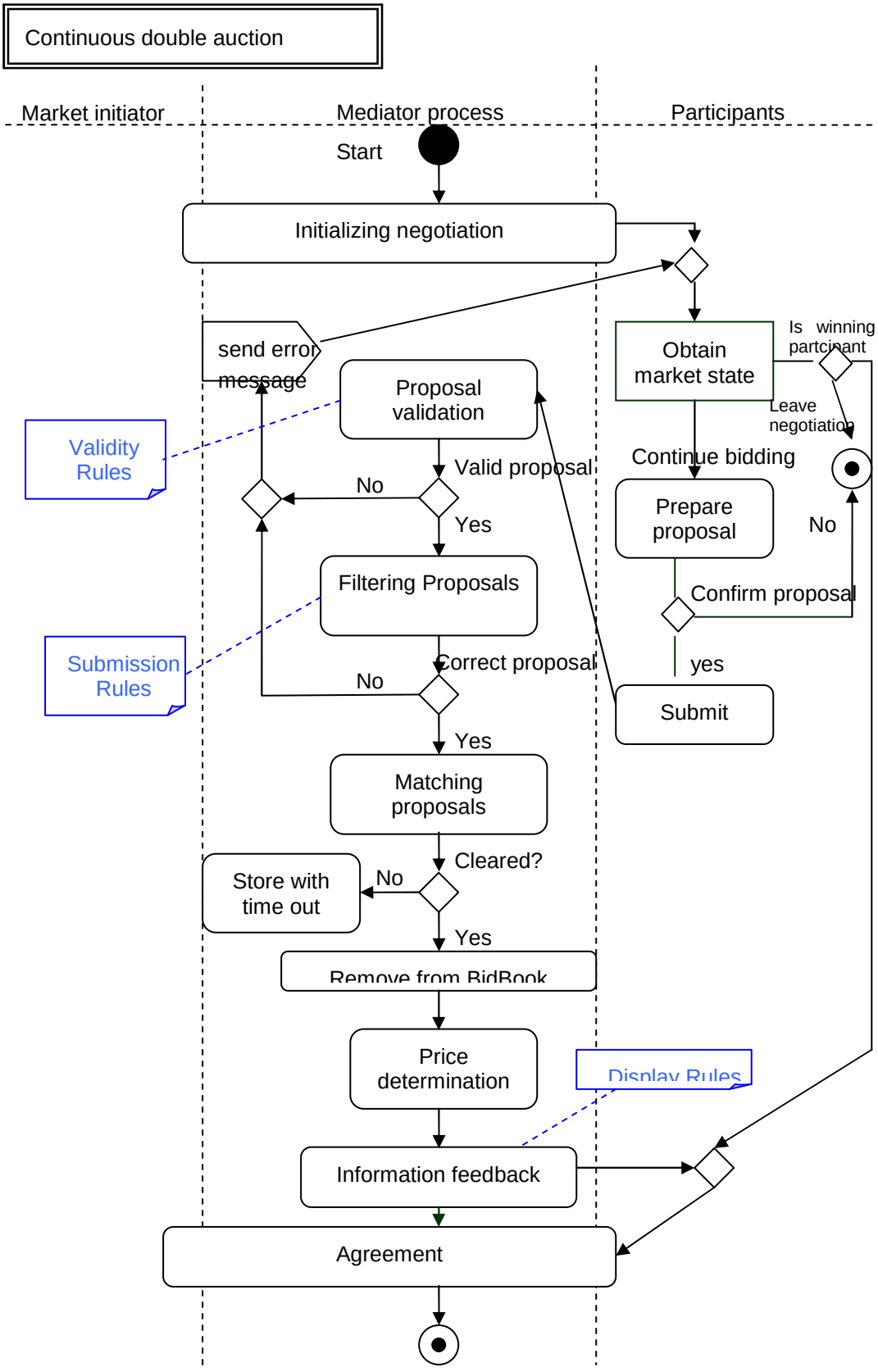


Figure 21 Activity diagram of a continuous double auction

An abstract market negotiation process may be characterized by the following parts:

The protocol represents the valid order for exchanging negotiation primitives of a given protocol such as the English auction, between participants. This is represented by the set of possible states (a state chart), possible transitions, and the firing rules. Each transition is guided by a rule that triggers an action or/and a message transmission.

Governing rules are applicable at different stages and stages of the negotiation process. Referring to the figure 20 and 21, each transition in the activity diagram is guided by a given rule. The set of rules can be categorised as:

- ✓ **Admission** rules give authorizations to allowed participants that have valid credentials. After that, the remaining negotiation rules are made available to all participants.
- ✓ **Validity** rules ensure that received proposals are acceptable in the current market session (for example, in respect to some domain values).
- ✓ **Protocol** specific rules control the different activities of the market process at different stages and also constrain the bid proposals that may be sent to the market.
    - o **Posting rules:** when a participant can send a proposal (according to the current state of the negotiation).
    - o **Improvement rules**: allow participants to determine the value of the new bid according to previous ones (supposing that participants have been informed about previous values).
    - o **Withdrawal rules**: specifies when a proposal have to be withdrawn (deadline, agreement, etc.).
    - o **Informing rules** determine when participants may view the bids and the when and how the participants can be notified that a bid has been submitted or an agreement has been reached.
    - o **Agreement rules**: according to proposals evolution, these rules determine how an agreement can be reached. The market terminology refers to these set of rules as *clearing* rules. These rules regulate how the market determines the allocations – that is, the winning sets of bids from sellers to buyers.
    - o **Termination rules**: determines the conditions of termination of the negotiation. For example, an English auction is terminated when there is no price increase on the traded item.

Negotiation Object: comprises the range of issues over which agreement needs to be reached. Within a market based negotiation system, this concerns the set of products that are traded and their negotiable attributes. Market protocols negotiate on the price (and also in fact on the quantity, since the price that has been agreed upon has an impact on the quantity desired by the participants).

Market stages

Market negotiation process involves three main steps:

Initialization or pre-negotiation: The initiator of a market, such as a supplier that needs to start a new auction to trade its CPU resources, specifies its negotiation objects (the offer). Participants (for exp. consumers) search for active market processes according to some matchmaking procedures. They then *Register* at the selected market.

Market negotiation: Participants exchange bids (proposals) through the mediating market negotiation process. The market rules govern the way that the exchange takes place – that is, contains constraints over some or all the parameters that are expressed in the bid proposal. This phase starts when the *negotiation object* has been initialized; that is, when the initiator has configured the market process with the initial offer. The market process may be accessed by participants through a minimal set of APIs that permit them to *submit* their bids to the mediator process, to *query* the current market situation, and to eventually *withdraw* a bid. This process will also use the APIs of the trading agents to *inform* them of current market status – that is, provide the feedback on the current prices at the market.

Post negotiation or agreement: This phase consists of establishing the agreement contracts between the winning participants (winner pairs of buyers and sellers). The agreement (contract) specifies the rights and dues of each participant, that is: the characteristics of the negotiation object, the payment features and the resource access process. When this phase starts depends on the structure and type of the market. In a continuously clearing market, bids from buyers and sellers are matched when possible – according to the matching rules of the market. For example, in continuous double auctions, an incoming bid from a buyer is immediately matched if it proposes a price that is higher than the lowest sell bid present at that time. Scheduled markets have specific clearing times, once the set of bids that have been received. In the latter case, the agreement phase starts once the market has cleared and determined all allocations.

3.6.4 Market Auctioneer framework

This section presents first the UML class diagram representing the main components of our market framework, and then we present a detailed outline of its functional components and their interfaces.

The class diagram in figure 22, describes the main activities of a market process. The trading process

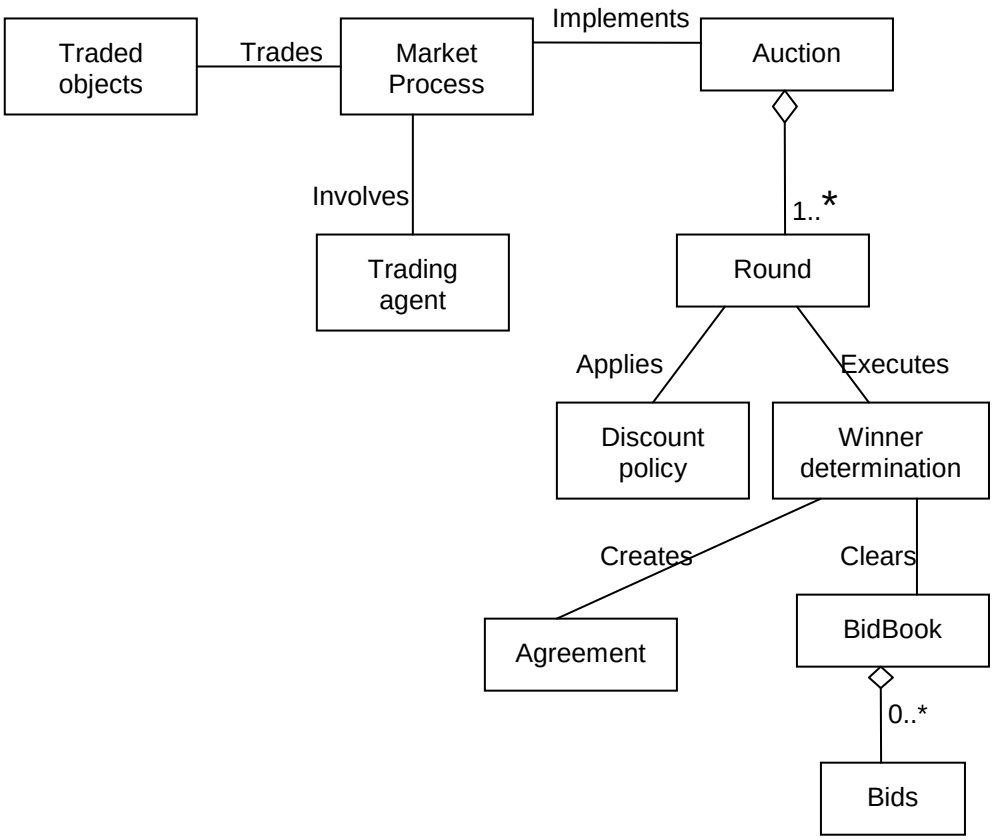


Figure 22 Class diagram of the market components

(*Market process*) is created by some *Trading agent* that wishes to buy or sell some resource (*Trading objects*) to/from another agent. The trading is guided by the rules of the negotiation mechanism encapsulated by an *Auction* that can be composed of one or more *Rounds*. During each round, the objective *Bids* are received and stored in a *BidBook*; at the end of the round the winner determination module determines the current set of winning allocations and the current prices of the resources. This module also determines if the stop conditions are attained. The final round calculates the prices to be paid are determined after applying a *Discount policy*. At the end of the market process, an *Agreement* object is created: these associates matching pairs of bids (from seller and buyer) that have won.

Functional components

The market framework (mediator process) is formed of four main building blocks (see Fig 23): the *Access control* responsible for the participants' admission to the market, the *Bid management* that receives and stores bids, the *Clearing* block that determines the winning bids and the prices to be paid, and the *Agreement* part responsible for building contracts between winning participants.

In addition to these components, we also introduced three new structures for storing the data handled in the market: bids, agreements and orders. These structures are:

- *BidBook* is responsible for storing the received bids and offers whatever the auction type.
- *AgreementBook* is responsible for storing the contracts of matching sellers and buyers; this is prepared on output from winner determination and price determination modules. According to the rules of the market, may need to implement agreement before passing to Order.
- *OrderBook*: in the context of Grid4All, an order is the final conclusion which should be executed. It results from the contract formed once buyers and sellers have agreed.

At the buyer agent side, we focus on its main "economic" task which is the bid specification. In an auction, participating agents specify bids according to some bidding language that takes into account the auction requirements. A bidder agent has to specify:

- *Resources*, by précising the list of resources it is bidding for and their attributes,
- *Time attributes*, by specifying the duration and the period of attribution he's looking for,
- *The price* that he's willing to pay for needed services,
- *Constraints* he needs to express on its bid.

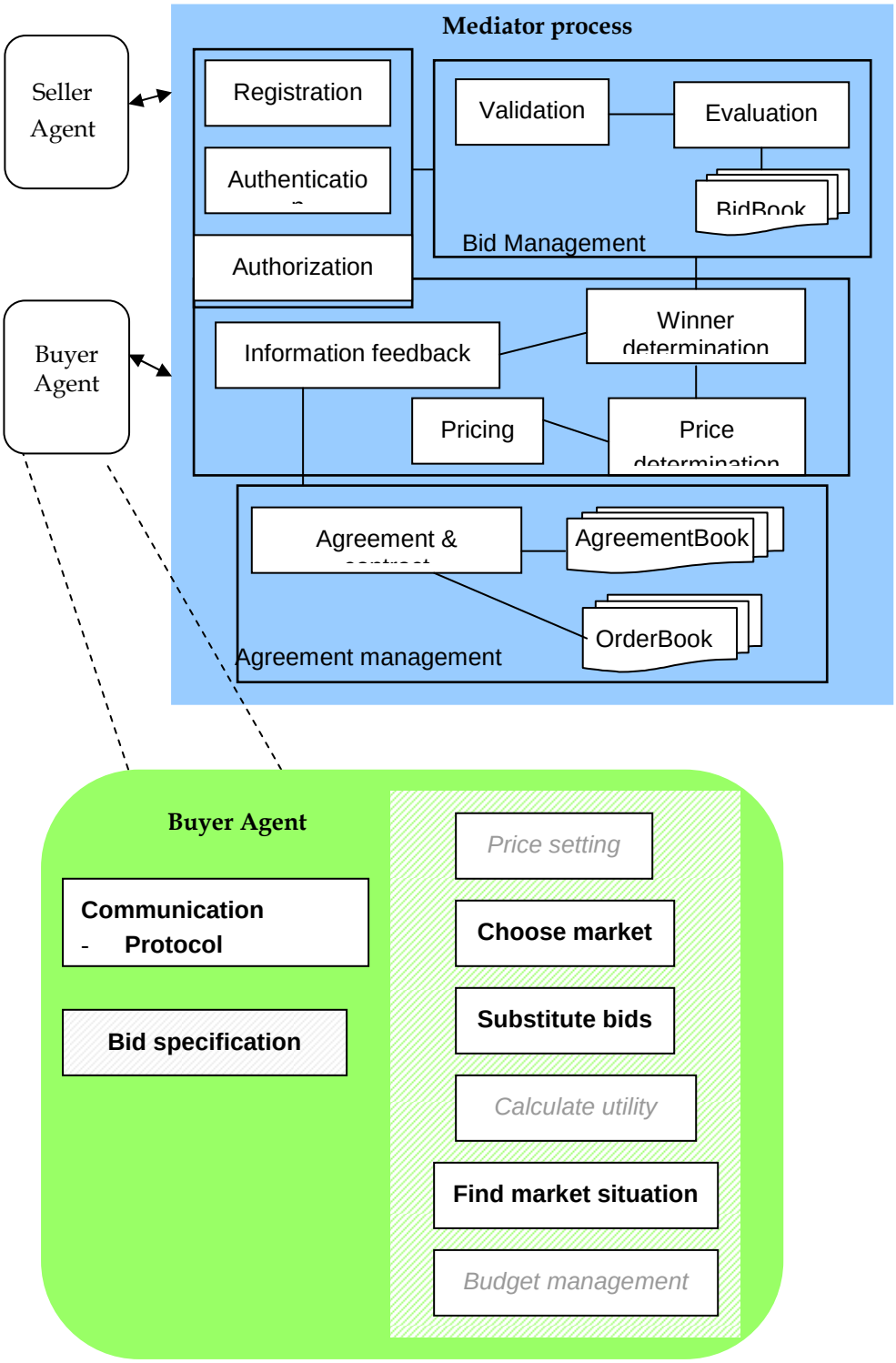


Figure 23 Functional components of market process

Components and interfaces

The components forming the market process reflect the main system and economic activities conducted from point of view of the initiator of the market and from the point of view of participants.

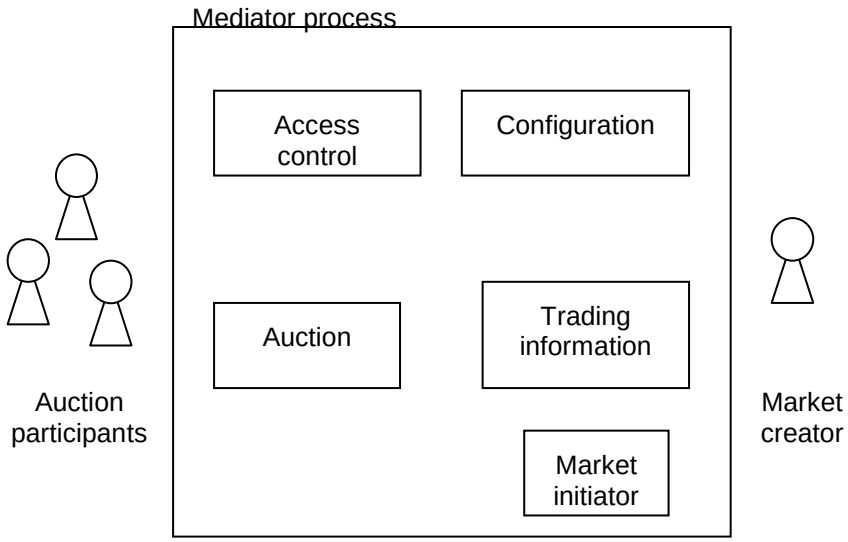


Figure 24 Main components and interfaces

The mediator process is composed of four main composite components (see fig 24): Registration and participants' management, Configuration, Auction Trading information and market creator. This implies several types of (external) interfaces of the mediator process:

Access control component manages the registration, authentication and authorization of market participants. This offers an interface *Registration* that allows a *Participant* to register. It also authenticates and authorizes the participants if necessary. The registration component also provides information about the registered participants through the *ParticipantManager* interface.

Configuration interfaces are mainly used to configure the market parameters (eg. market type, number of participants), the schedule parameters (eg. opening time, end of registration) and the market events (eg. market starting, bid reception). Three of the main configuration interfaces are: *MarketConfiguration* that allows the initiator to set the structure parameters of the market – such as maximum and minimum number of traders of a given role; the *MarketSchedule* interface sets the scheduling parameters such as the opening and closing times, the bid submission times, the clearing times; the *MarketListener* configures the

Trading information is a component that allows the initiators of the market to configure the type of resources that are to be traded within the market, their qualifying attributes (specific to resources) and also market related configuration such as the reservation prices (or maximum prices). It offers two main interfaces: *MarketQuote* and *ResourceInfo.* The MarketQuote interface allows the other components of the mediator process to update the current prices – on items or bundles. The ResourceInfo interface allows the initiator of the market to configure the set of traded resources and all related properties.

Auction interfaces encapsulate the auction specific configuration interfaces – these include both the activity control and also the parameterisation interfaces: *AuctionConfiguration* allows the initiator of the market to parameterize the auction and *AuctionRounds* allows the market process to control the auction activity cycle. Each round starts the pipe-line of the three main activities: bid reception, clearance, and information feed-back.

Taking advantage of the hierarchical component model proposed by Fractal, some market components are composite (such as the Auction component). In the remaining of this section, we will detail every composite

component and give its hierarchical decomposition. A graphic view of all the market framework Fractal components and their interfaces is presented in figure 25.

Access control

- ✓ **Registration**: registers participants that wish to participate in a given market.
- ✓ **Authentication**: checks according to admission rules if a new participant has valid credentials and thus it is an authorized trader for a given market.
- ✓ **Authorization**: verifies if a registered and authenticated participant is able to participate to the current market (max number of traders …).

Market

- ✓ **Trade information**: encapsulates the set of resources traded in the market and their attributes.
- ✓ **Bid Management**: A bid can involve an atomic resource, a composite resource or an aggregation of resources. The bid management component is responsible of filtering the incoming bids and preparing them to be processed in the market.
  - o **Pre-processing**: This component processes the initial bids of participants in order to make them exploitable by the market components. For example, if a participant formulates a bid that has not a primitive structure, it has to be processed in order to decompose it in primitive bids. A primitive bid is a set of bid attributes, such as the resource (a description the traded resources), the price (an amount and a currency describes the price proposed), Lease (a description of time lease of the resource).
  - o **Bid validation**: verifies if a proposal is (syntactically) well formed according to some validity rules. For example it verifies if the bid specifies the adequate transacted resources and units.
  - o **Bid evaluation**: verifies the semantic of the proposals according to bidding (improvement) rules.
- ✓ **Auction**:
  - o **Clearing**: Clearing component determines matching bids, calculates the price to pay and send this information to the concerned participants.
    - ▪ **Winner determination**: This component determines possible allocations according to the *Winner determination rules*.
    - ▪ **Price determination**: Price determination component decides what each buyer pays and what each seller is paid. This is based on *Price determination rules* of the used auction.
  - o **Discount**: This component calculates discount on prices paid by (or to) the participants
  - o **Information feedback**: Determines the type of information to publish (current price, winning price, winning participant ID, etc.), when these information have to be spread (according to *display rules*) and the participants that can receive it (according to *visibility rules*).

Agreement

- ✓ **Agreement builder**: creates an agreement between a bid and the satisfying asks. Agreements are created once the winner determination module has found the matching sets of bids from sellers and buyers. Formed Agreements are forwarded to the Agreement Manager, which is responsible for the generation of the agreement contracts.

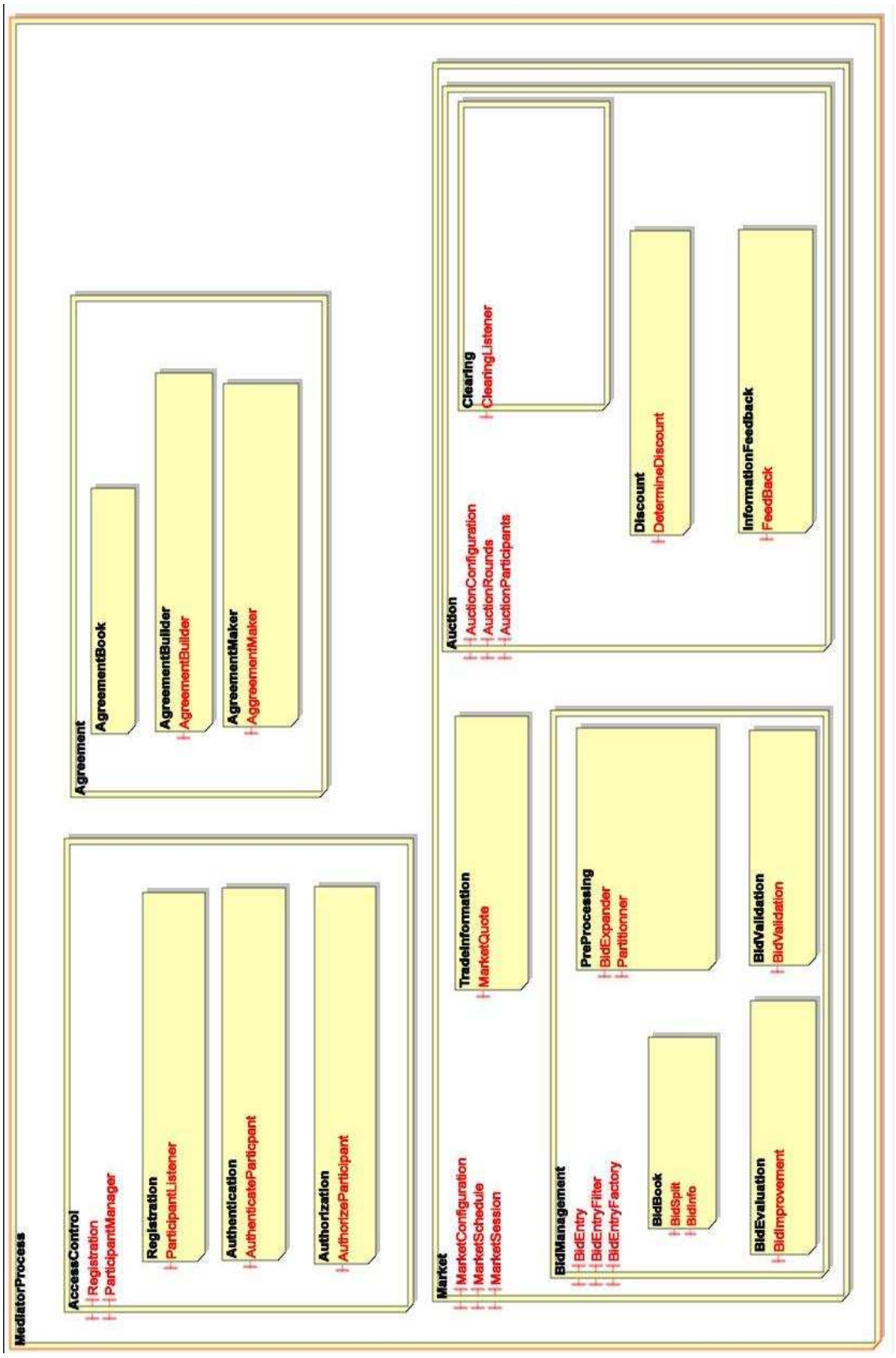


Figure 25 Market components in Fractal

System issues

So far we have mainly discussed the domain specific components. The framework needs to also address and provide abstractions for system specific concerns: activities, parameterisation and configuration, life-cycle management, and logging. The main issues that to address are:

- Composition and assembly

This assembly and creation of the market process through the composition of the coherent set of components that are needed to implement a given market protocol needs to be addressed. A straightforward approach consists of static specification through the Fractal ADL – the content classes furnishing the needed implementation needs to be described within the ADL which will then be deployed by the Jade framework described within Chapter 1. With this assumption, assembly may be looked upon as an off-line issue. Nevertheless (i) we need to validate this under different requirements to understand if there is a need to terminate the assembly at run-time and (ii) address linking of orthogonal components, as for example the winner determination and the discount components; the former does not impose a specific choice of discount policy.

- Configuration

This relates to the coherent propagation of configuration parameters to the components of an assembled and deployed market process. The ability to hide configuration details of sub-components, in the case where the sub-component assembly needs to be visible only at the level of the component is a desired feature. As an example, the k-double auction mechanism that has been presented in section 4 relies on a specific kind of sorting algorithm that permits an efficient implementation. Configuration of the sorting algorithm to the clearing sub-component preserves encapsulation.

- Life-cycle control and activity management

Life-cycle control consists of starting and stopping the execution of the components. This is related to activity management since components may be either passive or active. Active components in general have tasks that are executed – reception of bids triggers an activity to validate the bid and verify that it respects the bid submission rules. Passive components execute their functionality in the context of the calling component and may themselves invoke other component interfaces in the context of the invoking task. An example of a passive component is the *Solver* sub-component of *Winner Determination* that may be used to find trades that maximize the value of the market through the execution of an optimization algorithm implemented with a given optimizer. The Solver component drives the optimizer by building the model providing the variables of the optimization problem.

Market mediator process as a Web Service

Web services provide the means for software components to communicate with each other on the web. Within an electronic market place, there are three key roles: buyer, seller, and the mediator, which may in fact be deployed by participants in the negotiation. The last has been described in detail in this section. Market negotiation consists of interaction between these roles. We propose that the functionality of each of these roles be exposed as Web Services. The execution of the processes corresponding to these roles and in particular that of the mediator may then be modelled as a business process and specified using business process languages such as BPEL4WS; the resulting executable template can then be deployed within a business process engine. From the point of view of other roles, there is no difference between the business process and a Web Services. Messages that can be exchanged with the market process is described using XML schema – section 3.4 has presented the bid schema defined using XML; The Web Service that corresponds to the executable business process is described through a generated Web Services Description Language that provides a list of operations – such as *submit, withdraw, query*.

The components that have been described in the previous section may indeed be deployed as Web Services [SCA]. Specification of the market mediating process as a business process through the use of languages such as BPEL4WS amounts to describing work flows of a specific market negotiation protocol. Implementation of market mechanisms using BPEL4WS provides interoperable interfaces to interact with heterogeneous participants. Each process is exposed to the entities taking part in the negotiation process as generic Web Services APIs (bid, query, etc.) that correspond to negotiation messages.

Summarizing, service-oriented approach for modelling negotiation protocol has the following advantages:

- ✓ It enables advertising and using protocols seen as a web service. Hence, they can be deployed, located and invoked by a negotiation system.
- ✓ A web service orchestration language can be suitable to describe a negotiation process.

3.6.5 Agreement management

The last main component related to the framework is that of agreement management. The agreement manager has the responsibility in the surveillance, and enforcement of the implications of an agreement that may last for a potentially long time.

Following the negotiation, the market process determines the allocations, i.e., generates the winning pairs of buyers and sellers – this should be transformed in an agreement. The objective of the agreement management is the establishment of a bilateral contract between buyer and seller to: (1) ensure payment, and (2) provide a handle to access the resource involved.

The acceptance of an agreement implies a process, initiated by a call from the winner and price determination module (clearing component) denoted by call to *settleAgreement(handle)* to the **agreement manager**. The agreement manager receives a handle to that agreement and verifies that the winning seller and the initiating buyer accept the agreement by sending *acceptAgreement* messages to both the winner and the initiating buyer.

Here the agreement contract must be established based on the agreement decision reached at the market process, and must be enforced. The agreement contact includes:

- Creation of a lease generated by the supplier agent with the *capability* to access the resources.
- Logging the transaction for auditing purposes (to keep a non-repudiable log of agreements).
- Payment process that can be done before, during or after the use of the goods, and as a result a proof of payment has to be provided in the form of a payment receipt.

3.6.6 Market protocol factory and repository

This section discusses the rationale, requirements, and the preliminary approach to design the market protocol factory and repository. A first pre-requisite to the implementation of the repository is the adoption of methodologies and tools to specify market negotiation protocols. Negotiations are mechanisms that allow a recursive interaction between a principal and a respondent in the resolution of a deal [OMG99]. Negotiations proceed by allowing each of the parties involved to make proposals that are beneficial to themselves and allowing them to incrementally revise their proposals in order to come to an agreement. In order to allow consumers and suppliers to participate in different types of markets (that may implement different market negotiation mechanisms), the following key requirements need to be met:

- Shared understanding of the interaction protocol and market rules,
- Common definition of the exchanged messages,
- Executable negotiation processes that can be deployed and configured by the participants.

Problem statement

Current approaches for the construction of Grid resource markets mainly consist of operating central market servers per segment of Grid resource or service -- this central server may implement one (or a few) type of market protocol. This is similar to the operation of the eBay auction server. Sellers (or buyers) needing to sell or buy some resources contact this central server to initiate a new auction instance – they choose one of the protocols implemented at the server. Agents desiring to participate at this auction are assumed to know and agree to the implemented protocols.

An open and dynamic environment with different target market segments cannot rely on a single and immutable negotiation protocol. The environmental factors such as potential number of participants, their preferences, their urgencies, the types of resources offered/required, their characteristics, and the endowment of the participants, are factors that affect the choice of a market protocol [KSS00]. [PCr03]

summarizes this as follows: "Good market design begins with a thorough understanding of the market participants, their incentives, and the economic problem that the market is trying to solve".

Recent years have seen a number of attempts to design configurable multi-protocol electronic market places. *AuctionBot* [WWW98] supports the configuration of various auctions; *eMediator*[SAND02-1] allows participants to setup markets – it offers a wide range of auction mechanisms to choose from and supports the user in choosing an appropriate market type. [KLS04] report the design of a configurable negotiation server that supports bargaining, based on a process model that organizes negotiation activities into phases; and a set of rules that govern the processing, decision-making or strategy, and communication. [MS01] presents the *Silkroad* platform designed to support multiple auction protocols.

A protocol may be seen to be a set of rules that participants need to respect during a *conversation* – the process that proceeds according to the rules of a given protocol. The rules determine who may take part within a conversation and how each participant must contribute to its processing. [SRT05] defines choreography as a description of the peer to peer externally observable interactions that exist between services. The interactions are described from a global point of view and not from the perspective of a particular role. Choreography description can be used to generate the necessary behavioural contract for each of the roles and to verify that all the separate cases are consistent with each other. Once the choreography definition is created, machine-readable executable forms need to be generated for each role that may participate within the choreography such that each role may enact its individual process.

Requirements

The component architecture and framework for the developing of market negotiation processes that is presented in section 3.6 facilitates the design of market mechanisms and trading agents. This by itself is not sufficient – a protocol designer is also concerned by the practical use of the protocols by the participants so as to strengthen the interoperability between the participants that assume different roles. In the following we identify the following main roles:

- The auctioneering or the **mediating market process** whose responsibility is to initiate a market session and to collect the bids (proposals) from the participants, that is, the buyers and sellers.
- The **initiator** of the mediating process is a special participant, who based on internal decisions decides to and starts a market process. In practice the initiator is typically either a
  - Seller who desires to start a new market to trade its resources in a forward market
  - Buyer who desires to start a new market to acquire resources. This can be looked upon as a Request For Quotations (RFQ)
  - 3rd party mediator that decides to initiate a market to intermediate between a set of buyers or sellers
- The participant which may be either a **buyer or seller** (or both in the case of exchange markets), whose objective is to send their bids to the market process and to achieve trade(either buy or sell)

Subsidiary roles are played by other services implementing agreement management, bank, payment, logging etc. Nevertheless for the purposes of discussion it is sufficient to focus on the three main roles.

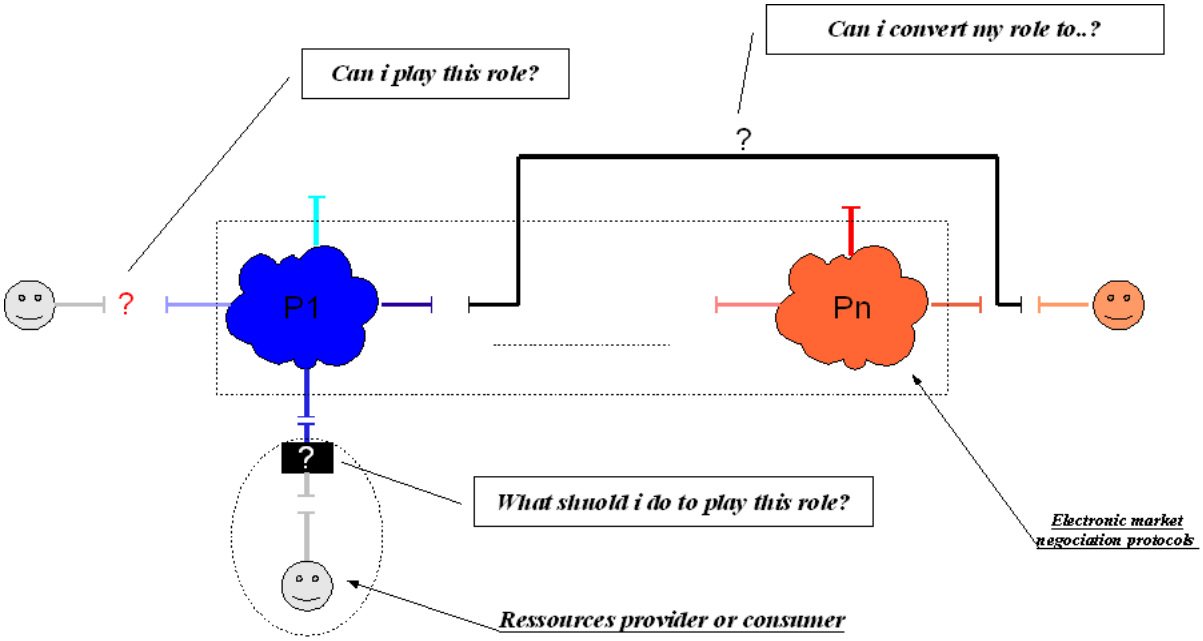
Participant point of view: Implementing auction mechanisms in an open context requires tools to aid participants to:

- Find an appropriate auction mechanism
- Obtain the rules and the underlying protocol so as to allow the interaction with the market (auction) negotiation protocol

The market factory acts as a repository of market protocols that stores a library of developed protocols in an executable format (or that may be translated on-line to an executable format). A participant, i.e. a buyer, seller, or a 3rd party broker wishing to start a new market session uses the services of this repository to choose the executable template of the appropriate market protocol and to instantiate the template corresponding to that of the mediating market process. It is then assumed that this new market process is advertised within the information service such that participants may be informed of its existence. To take part

at a market buyers or sellers need to be able to interact with this auctioneer process – this requires that the participants be doted with an interactive behaviour that is conformant to the specification given to the role of seller or buyer with respect to that of the auctioneering process.

In an open world, functional and semantic heterogeneity persist. It is unrealistic to suppose that market actors (resource provider and consumer) define from scratch (while instantiation of a protocol role) their behaviours each time they are involved in a negotiation. As discussed in previous sections, it is also unrealistic to restrict actors to implement a specific negotiation protocol. Being developed separately there is no reason to assume that any two given actors use the same (i) negotiation protocol (ii) the same design of a given protocol (iii) abstraction level while implementing roles – and especially for the consumer agents as is resumed in the figure below.



Here the question is: how to design the structure and behaviour of the participating roles?

Designer point of view: The second aspect is that from the point of view of development of new market protocols -- developing a new auction based negotiation mechanism should not result in a software developer having to write the program code entirely by hand – this is long and tedious and error-prone. Ideally a designer should be presented with a protocol modelling tool that facilitates the design of negotiation protocols. Pre-defined negotiation templates may aid in the reduction of design time – the developer need only concentrate on the aspects of design of the market protocol, in particular the rules (and required algorithms) of the market clearing mechanisms, the bid submission, and the termination rules.

The questions to answer here are:

- *How to help the designer produce the interaction protocols given a high abstract and formally defined process model.*
- *How to automatically derive the behaviours of each role involved in the interaction*

System point of view: [CASS] suggests that an automated negotiation must have four necessary properties. It must be (1) Correct: the negotiation must not deadlock or incorrectly handle exceptions (2) Reasonable: it must allow for uncertainties of the wide area network (3) Robust: it should continue to function

despite improper action by the bidders and (4) Fast: it has to execute and also converge quickly. [BEN04] also argues that an automated negotiation should also be traceable.

Achieving these five properties necessitate a formal approach towards the specification of negotiation and market protocols. As [CASS] says "notation with a well-defined, well-formed semantics can facilitate verification of desirable properties".

Formal specification of interaction protocols allow for the analysis of desirable properties such as safety and absence of deadlocks. A safe and sound interaction allows no unpredictable states and transitions -- negotiation states that become valid are only those that are defined by the protocol. Proving of live-ness property ensures that a process will eventually enter a desirable state and that there will be no deadlocks. A protocol can be ensured to terminate if it can be proved that all paths in a protocol lead to a terminal state. [SPB06] discusses the importance of termination rules in iterative auctions. They give an illustration of an example case of bidding where the stopping rules of the RAD design [KWA05] is not sufficient and may lead to looping. This behaviour is due to the fact that linear prices that are calculated (at the end of each round of bid submission) are not required to be monotonic.

A negotiation must either eventually terminate after a process or the entire negotiation halts in a state in which execution of a finite number of actions complete the negotiation. Hence accessorily formal analysis of protocols may help in the determination of the number of iterations before which the negotiation will terminate – this feature may be helpful in the design of the parameters of the market/negotiation activity and bidding rules.

A protocol defines the rules for a conversation and the rules of a chosen protocol have to be conveyed to each participant. For example, participants need to be informed that the rules of bidding within an English auction constrain a bid to be superior to the current price by an improvement imposed by the initiator of the auction. A well defined protocol helps participants to abide by the rules and make progress towards the consensus.

The questions that are asked here are:

- *How to verify properties of the specified negotiation protocol?*
- *How to validate that a given set of participants are indeed able to enter into a valid negotiation conversation?*

Now to meet the requirements generated from the three points of view within the landscape of service oriented negotiation architecture, we need to:

- *Address capture of behaviour of BPEL in a formal way, since even though BPEL is useful being an executable language, it does less well as a language for modelling abstract interactions* [VDA05].

Conclusion and approach for future work

Market based negotiation systems may be approached from a business process perspective – negotiation is an inter-organizational process where each of the participants executes internal and external 'business' processes to reach a business objective. The Service Oriented Architecture (SOA) approach is valid within the e-market place -- market services may be decomposed into elementary services at a lower level to foster a wide and large number of implementations. The Web Services and WS-based process definition standards provide mechanisms for defining negotiation processes that can be deployed in a platform-independent manner.

In SOA, the (business) process view deals with the description of the interactions in which a given service can engage with other services, as well as the internal steps between these interactions. BPEL4WS has acquired the momentum to become the Web Services-based process definition standard. Section 5 has discussed the reasons for which we address the definition of a negotiation process by using BPEL4WS. However service interactions go beyond simple sequences of requests and responses or involve large number of participants. Composition deals with the interplay between services and business processes. Choreography captures collaborative processes involving multiple services where the interactions between

these services are seen from a global perspective, by describing the collaborations between the services in order to achieve a common goal. A *choreography model* captures the interactions in which the participating services engage in and the dependencies between these interactions, including those of control-flow (which interaction must occur before what other?). All partners are treated equally and the interactions are captured from a global perspective.

In the terminology of Web Services, the specification of negotiation protocol corresponds to the definition of the choreography of the system, expressed in a specification such as WS-CDL. The processes implementing this choreography that is implemented by each of the roles within the negotiation is defined using a workflow language. BPEL4WS supports design and implementation of WS-based processes by providing mechanisms to preserve the state of WS by correlating messages exchanged between WS. BPEL4WS has the expressive power to support commonly required workflow patterns [WOH03].

From a choreography model, the behavioural interface (for example in BPEL4WS) that each participating service must provide within a given collaboration may be generated. This behaviour interface may be used to develop the service – for example to generate the interface that a buyer agent must provide in order to participate within a choreography describing a given auction protocol.

Even by considering the use of Web services language WS-CDL and BPEL to describe the actors published protocols heterogeneity problems persist in two specific levels: the first is commonly called functional heterogeneity, for example is the Grid4all project two actors can exhibit two different roles of two different negotiation protocols. The second called semantic heterogeneity where in two different protocols we make two different references to the same concept for example a bid. This heterogeneity persists due to the symbolic nature of the description language and the lack of a common semantic reference. Hence the **two main challenges** that we need to address are:

- Role compatibilities: given a choreography model of a negotiation protocol specification described using WS-CDL. If we suppose a set of actors: buyers and suppliers where each defines its own negotiation protocol using a BPEL interface. It's important to hold a method that allow to each participant to check the compatibility of it protocol with a specific role of the chosen WS-CDL protocol specification.
- Partner behaviour adaptation: In case of behaviours incompatibilities to a given protocol, methods must be established for an automatic role instantiation. Generating from WS-CDL protocol specification a BPEL interface for a specific role. Also we need an automatic method to generate models for wrapper that adapt a predefined participant negotiation protocol to a specific role of a negotiation protocol.

WS-CDL is a programming language oriented towards well-structured communication-centered programming due to its notion of channel types. The choice of WS-CDL is also motivated by the fact that this choreography language may be considered as an implementation of the pi-calculus, a message-based formalism that is suitable for the analysis of global perspectives of distributed protocols

The W3C community is very active around WS-CDL both at the formal consideration and also at the technological level. As has been the case for WS-BPEL, we expect that WS-CDL will be largely adopted as Internet global communication centered languages – WS-CDL initiative is backed by vendors such as Oracle Corporation, Sun Microsystems etc.

At the first step, our plans are to evaluate the tools developed by **PI4SOA (www.pi4tech.org)** which is the W3C group implementation of WS-CDL. The design and development suite provides a WS-CDL editor that allows designers to create the choreography model and PI4SOA development tools enable validation of the choreography description and generate projections of endpoint descriptions. The generated Endpoint called also skeleton (for each participant) can be either Java code deployable using the Apache Axis or directly as BPEL services using orchestration engines such as ActiveBPEL.

3.7 Services for the market place

Two main services in the market place, market information and currency management, are described in this section. The market information service provides market relevant information to market participants. The currency management provides a payment service for market agreements and accounts for market participants.

3.7.1 Market information service

The marketplace system, as it has been defined in previous sections, is considered to be a decentralized, dynamic and large in terms of distance and number of participants where distributing and obtaining information can be prohibitively costly in terms of traffic and coordination. However, in a changing and complex environment, agents need information about status and changes to effectively decide. This implies obtaining information that can be derived from a few to many entities at a reasonable cost. That information can be detailed or may be aggregated to reduce the effort and cost of distribution. Entities may publish information without having to consider which entities might be interested. This leads to the need of a service that mediates, provides indirection and distributed processing (including aggregation, summarization and other functions) and efficiently routes information between sources and sinks of information. The market participants, a buyer or seller agent (as defined in section 3.4 and 3.6), could then express interest on some data that might derive from a large amount of data (e.g. related to a global or regional status). If this information is progressively aggregated and processed the volume of information is efficiently reduced and also rendered anonymous. Cost of circulating information is dramatically reduced and the sinks of information can obtain just the right data with a very low cost about any range of entities, event being able to provide information on the global status.

The information managed by the market participants falls into two categories:

1. **Static information** describing resource characteristics which have a very low rate of change. Examples: CPU frequency, OS version, memory size, etc.
2. **Dynamic information** which have a relatively high degree of dynamism. Examples: price, load, availability, performance, etc.

The market agents should be able to publish events each time their internal state changes (price, load, availability, etc.) or to subscribe for events they are interested in. These exchanged messages consist of attribute-value pairs. Consumers of information should also be allowed to express filtering constraints over the attributes of the published messages. The system requires the ability to express aggregated queries by the use of any SQL-like aggregation function such as MIN, MAX, SUM, COUNT, and AVG.

Queries on index (one-shot-pull):

a) Return the minimum price of seller offers for the good. Example: “CPU time on x86 Linux machines whose load lies within the interval [0.5,1.0]”

b) Return the maximum price of buyer bids for the good. Example: “CPU time on x86 Linux machines belonging to the domain /Grid4All/X/Y/Z”

c) Return the average price of seller’s offers for the good. Example: “digital photo processing whose reliability is greater than 0.8 and which are located in domain /Grid4All/Y”

Notification (subscription):

d) “Notify me any time a new seller’s offer is posted for good X whose price < 60€ per minute”

e) “Notify me when the maximum price of bids for the market named ‘Auction86798’ changes”

f) “Notify me when there is an offer for the good ‘CPU time on x86 Linux machines’, only from sources at a distance less than D from me.”

The API for the MIS provides the following four methods to get the information of the economic

requirements:

public void publish(String topic, Object info)

The aim is to make an information object public to the market. Primarily this is the publishing of information to a group of subscribers with the same topic. But it is also the disseminating of information within the aggregation mechanisms.

public Object getSummary(String topic, Function function)

This method corresponds to the query on a index. It returns a summary of information belonging to a certain topic. The data can be aggregated by passing a SQL-like function like “SUM where periods < x”.

public void subscribe(Subscriber subscriber, EventListener listener,
String topic, Function function, long frequency)

Subscribes to a topic to receive updates. The subscriber is notified through the EventListener, when a new event occures concerning the defined topic.

public void unsubscribe(Subscriber subscriber, String topic,
Function function)

Removes the subscription of a client to a topic or content. For example, when it needs no further information about new products in the market.

Object[] getHistory(String topic,long init, long periods, Function function)

An array with summaries occurred on different time is returned. The time is defined by the number of last periods.

State of art

In order to refine the design of the MIS the literature was explored. There are several recent research studies on economic information acquisition in auctions, as surveyed in [Berg06]. These studies are mainly based on theoretical expectations of information acquisition and less on technical implementations. Nevertheless, they show in a theoretic and analytical way the demand for disclosure of economic information in marketplaces.

From the high level of node churn or failures in a distributed on-line marketplace follows the need for a robust system. Actually, there are proven solutions providing a structured overlay network based on Distributed Hash Tables (DHTs). These mechanisms scale with the number of nodes and cope well with the churn without increasing significantly the network overload.

Publish-subscribe paradigm provides a loosely-couple interaction mode for notification of events in a large scale environment. Subscribers register to a topic of their interest or to a pattern of events, afterwards they will receive asynchronously information matching their requests. The strength of this paradigm is the full decoupling in time, space and communication between subscribers and publishers. There are many distributed publish-subscribe systems varying widely in their functions and structures. The MIS requires a content-based subscription model.

Data aggregation is indispensable in large scale systems to handle the volume of information, which grows exponentially. Moreover, aggregation services allow querying an index considering basic functionalities in distributed systems like load balancing. The aggregation abstraction combines the necessary information from various sources in order to provide a coarse-grained summary of the required information without having to be aware of every detail. The gathering can be based on topics or patterns, which are already applied in publish-subscribe systems.

Research in aggregation of events and dissemination of aggregated information is still immature. Few publish-subscribe based systems implement aggregation. Willow [Rene04] proposes an aggregation system with publish-subscribe model combined in one DHT-based protocol. Another approach for data aggregation is the Scalable Distributed Information Management System (SDMIS) [Yala04].

Requirements for economic information acquisition will be met by the MIS combining different existing technologies. The aggregation mechanism provides an efficient summary of the market and its participants. Time-sensitive market data will be obtained by the publish-subscribe model. Building these two approaches on top of a DHT-based structured overlay network, allows coping with technical challenges of the requirements for the distributed economic information system.

3.7.2 Currency management

In this section we define the currency management used within Grid4All in terms of the concepts explained in earlier sections. Grid4All will have ***a unique virtual currency*** (*g-currency*) that serves as a medium of exchange between all trading agents (buyers and sellers) managed by the currency system. This assumption simplifies currency management and does not imply any loss of functionality. Any transfer outside will require exchanging to real money.

For simplicity, efficiency and for security reasons we implement the currency as an ***account balance-based system***. This means that there is a trusted entity that manages user accounts. Every transaction should use this trusted service in order to carry out the transaction. GRIMP has unique service that mints and sells specific g-currency.

User accounts are kept in a central banking service. The word central should be understood as administratively centralized. The implementation of this service is distributed over multiple nodes organized organize as an overlay using a DHT (*Distributed Hash Table*) in a way that each node is responsible for a subset of accounts. This helps to balance loads and renders the service reliable and dependable. All nodes forming the central service should be considered trusted.

This banking service will be the responsible to store a *log* with the different transactions in order to solve disputes between traders. The resolution of the dispute might not be done automatically and might need a third entity capable to solve it. The transaction protocol implemented is an on-line payment protocol, that is, users are authenticated and identified before transactions are accepted.

Finally, the Grid4All banking service will provide a general enough API in order to allow different kind of transaction orders as for example *pay before*, *during* or *after use.*

Currency market definition

The decision of having a banking service for managing accounts and payments is not bound to the *currency market* that will be employed. That is, this is independent to how currency is issued and managed from the economical point of view. There are two approaches as described in previous sections: a *closed currency market* and an *open currency market*.

The *closed currency market* can be defined as the market where there is only one currency and the direct exchange real currency against virtual currency is not allowed. Therefore, it arises all the problems related to managing a real currency (i.e. inflation, deflation, interest rates, etc.). The currency management system should provide mechanisms to control the amount of circulating currencies in order to maintain a stable relationship between circulating currency and traded goods. Therefore some *policies* should be defined to drive currency to equilibrium and the exchange between real and virtual currencies.

On the other hand, the *open currency market* can be defined as the market where the virtual currency is only an intermediate representation for real currency. In this sense, a fixed exchange rate can be performed (against the floating one of the closed currency market) and the banking service should only be used as an intermediate payment service without incurring with a real banking payment in each Grid4All transaction. This definition allows the currency management system to delegate all the problems related to currency management to the real institutions which manages real currency.

Security issues

As long as a customer needs to maintain an account with the banking service to perform payments, an authentication and authorization mechanism is needed to avoid impersonation as well as counterfeiting. Therefore, the banking service needs mechanisms to issue *non-repudiable documents* to report to both customer and supplier that a transaction has been made. This could allow users to perform reclamations against the other part of the transaction if the agreement is broken in payment terms.

Requirements for the Banking Service

In this section we introduce preliminary requirements for the Grid4All Banking Service. The banking service is a key component in market transactions and needs to be robust and reliable.

The goal is to provide a dependable component on which other components (such as the payment component) can rely. To achieve the previous mentioned properties our choice is to deploy the banking service on top of a DHT. This way the banking service can take profit of the self-* properties provided by the DHT and therefore avoid any single point of failure. We assume that peers are trusted and there is a DHT private to the banking service. Otherwise, there are additional security issues (e.g. privacy, tamper resistance, non-repudiation) that will require signing and encryption of the stored data.

Current DHT implementations suffer from the lack of a consistency model and support for *mutable consistent data*. Furthermore, banking and payment operations need *transactional semantics* in order to provide consistent results. For example, the operation *transferFunds* implies the atomic modification of two accounts (query, modify and save) in order to avoid inconsistencies in user account balances when concurrent modifications against the same account are executed. A transaction should assure that either both accounts are modified or none at all.

The key concept to achieve mutable consistent data is that no more than one peer should be responsible for a single identifier at a certain time (also called *lookup consistency*). Despite it is not a widely required property from DHTs, the overlay network infrastructure in Grid4All, namely DKS, guarantees this property. Considering a DHT infrastructure which solves the consistent data issue, solving the second requirement (ACID transactions) can be achieved *locking the object* and *serializing* different operations against the same object in its object responsible node.

The next figure shows an overall architectural view of the different components of the Currency Management System, the interactions between the clients and the banking service through the CMS API and overall view of different layers of a node belonging to the currency management network.

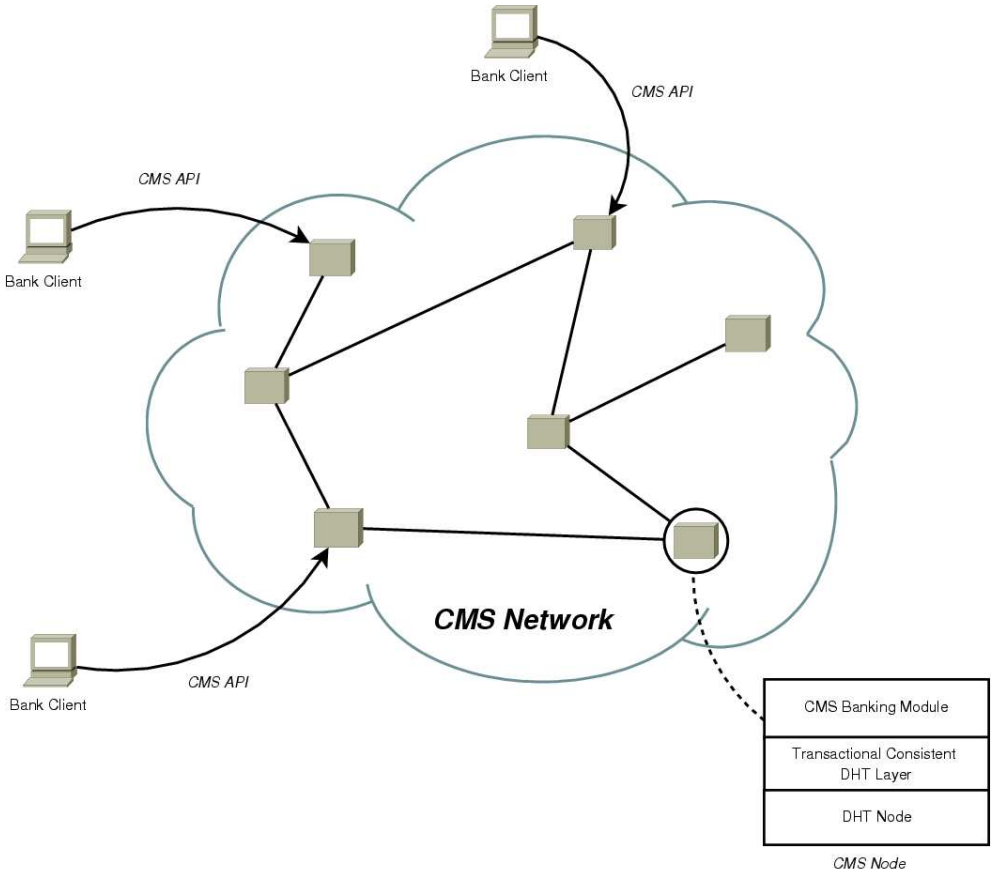


Figure 26 Architectural view of the Currency Management System

Banking service API

This section presents the current API for the banking service which is capable of managing accounts (creation and deletion) as well as operations to transfer funds from one account to another one. All the returned *Receipts* are unforgeable objects by means of some security mechanisms still to decide.

`AccountID openAccount(Credentials user)`

Create a new account and relate it with the given Grid4All credentials. It returns an AccountID specifying its ID which will be unique in the system.

`CancelAccountReceipt closeAccount(AccountID id, Credentials user)`

Close and delete the account from the bank system given an identity within Grid4All and an AccountID.

`Account queryAccount(AccountID account, Credentials user)`

Returns the associated account with a given AccountID and an identity within Grid4All. This account maintains a log with all the transactions performed within this account as well as the balance and reserved balance at query time.

```
TransferReceipt transferFunds(AccountID source,
                              AccountID dest,
                              int amount,
```

Credentials user)

Transfer an amount of currency from the source account to the destination account. This operation is atomic in the sense that both accounts are modified or none of them.

ReserveReceipt reserveCurrency(AccountID source,
AccountID dest,
int amount,
Credentials user)

Allow the owner of an account to reserve the amount specified for a future payment against the destination amount. This operation does not imply a real transfer of funds since the destination account is not modified. This method assures that the reserved funds cannot be spent in another transaction.

TransferReceipt commitReservation(ReserveReceipt receipt,
Credentials user)

Finishes the reservation of funds transferring the reserved funds specified in the ReserveReceipt to the provider's account specified in the receipt.

CancelPaymentReceipt cancelReservation(ReserveReceipt receipt,
Credentials user)

Cancels the reservation of funds made before due to some reason. Once the reservation is cancelled the funds are again available for spending.

3.8 Relation with other work packages and tasks

3.8.1 Semantic information service

In recent years it has become clear that there is overlap between grid computing and the benefits of a service oriented architecture based on Web Services [CZA04] [FK02]; services provide virtualization of resources and also that of grid functionality. We have in earlier sections presented the resource market as trading in resource services, where a resource service is a computing resource that encapsulates computational capabilities as a service.

Section 3.6 has presented our approach where a market service is represented as a Web service; that is, market mechanisms such as auctions are developed as services and are executed as a business process. The service interface of the market is represented through a WSDL comprising of a list of operations and messages that these operations accept and return. Traders discover the advertised markets and they access the market using the exported Web Service operations. Traders select markets based on both the characteristics of the market and also on the characteristics of the Grid services that are being traded at the market, that is, the services being *offered* (*requested)* at a forward (reverse) market should match the *bid* that the trader needs to be satisfied.

In an open market place, a meaningful matchmaking of *orders* (offers/requests) should be realized to allow matching of services based on their semantics instead of their syntactical representation; for example, a request for computational resource may specify a quality of service measure expressed in FLOPS (floating point instructions per second) that has been measured using the *Linpack* benchmark. A meaningful matchmaking service should find compatible resources whose rate of work capacity has been measured with equivalent benchmarks. Grid services may also have complementarities, meaning that they may need to be traded as a *composite* service. Matching of requests for {CPU, Storage} should select markets trading {CPU, Storage} and not otherwise.

Realizing of matchmaking in an open market place requires rich knowledge representations for services, capabilities, qualities. We are closely working with the task 3 (of work package 2) so as to provide the requirements for the design of ontology for Grid services and markets – so as to provide a shared ontology for the basic vocabulary that the participants and the market place must share. In particular our aim was to enrich the concepts of *offer, bid,* and *request* in order to enrich their expressiveness required by real-world constraints. The chapter 3 describes in detail the current status of work within Grid4All. Future work will investigate the possibility of extensions to the market ontology so as to be able to represent a set of useful economic properties.

3.8.2 VO management framework

Section 3.6 has presented the architecture of the Grid4All market place and section 3.5 has argued for a model of markets for Grid resources that addresses scale -- creation of 'local' spontaneous markets naturally tends to segregate the space of market participants. Section 7 has also addressed the establishment and management of agreement contracts that bind the provider and consumer of resources.

In this section we address two issues:

- Where will the services Grid4All market place be executed and where will a newly instantiated mediator (market) process execute?
- How will the virtual organisations that allocate resources at the market place access and obtain use of the resources?

Resource execution management

As has been described in the Chapter 1 on the framework for autonomic management of virtual organisations, one of the principal issues that are addressed within Grid4All is that of deployment – that is, the installation of application software, its configuration, and finally manages the life-cycle of the software components forming the application. At the fabric layer, the physical computing nodes forming a Grid4All virtual organisation are organized as a peer-to-peer overlay using technology provided by WP1. This overlay provides messaging and communication primitives and also serves to implement the functions -- discovery of resources, monitoring, fault detection that are required for autonomic management.

We envisage a utility model for virtual organisations where newly acquired resources are 'integrated' seamlessly into this management overlay. That is, resources are *leased* from the market by a virtual organisation for some time interval – the lease is a contract between a provider and the virtual organisation. During the period of lease, the resources are configured such that they belong within the target virtual organisation.

Minimal software set needs to be installed on the target nodes such that the following two actions may be executed:

- **Join**: Nodes that are allocated to execute applications and services for a given virtual organisation will **join** the underlying management overlay. They use the API of the overlay services provided by WP1 to effect this operation.
  - The pre-installed software is required for the join operation.
- **Provision**: The management framework detects a newly joined node and deploys the needed application software on this node.
  - The pre-installed software permits the deployment of new application components specific to the hosting virtual organisation, on the node through a basic set of operations: **install**, **uninstall**, **start**, and **stop**.

A key issue that will be addressed within the VO framework is that of *lease* management. Compute (or storage) resources are allocated at the Grid4All market for determined periods of time. VO management needs to ensure that the resources acquired through the market will be freed at the end of the period of use – unless the lease has been upgraded through further negotiation at the market.

Deployment model

The second issue that needs to be addressed is that of deployment:

- Of the software components implementing Grid4All market place services

– Of the mediating market process that is created on-demand by a participant acting as the initiator

We believe that the Jade management framework provided by T2.1 of this work package satisfies this purpose. The deployment model that we envisage is that of considering that the market place services execute over a peer-to-peer overlay that is managed by Jade (extended to peer-to-peer overlay based fabrics within Grid4All). This allows the deployment of software components of distributed applications on to a set of nodes – the Architecture Description Language furnished by T2.1 allows describing the different market place components and their bindings. The exact deployment architecture will be the subject for future work.

One of the services offered by the market place is that of selection of and deployment of a market process (as a mediator for market negotiations). A key issue that needs to be addressed is that of the selection of a node within the overlay on which the mediator will be instantiated. The choice of the node (or at least its characteristics such as location etc) may either be made by the initiator or may be subject to policies of the market place. We will address this in the future work.

3.9 Conclusions

The key issue of this task is the identification of requirements to implement markets for overlay based *neighbourhood* Grids.

We have focused on Grid resource markets and the architecture to design and develop open market places for trading of Grid resources. One consequence of open market places is the need to provide support for both designers of market protocols and for participants at the market place. This will be realized through the market factory or repository.

We have taken the approach to define the markets in a structured way; this is a requirement to match mechanisms to application scenarios and to provide methodologies and tools to designers of market negotiation protocols.

Future

The next steps consist of terminating the specification of the component framework for mediated market processes and to demonstrate proof-of-concept implementations of three types of market mechanisms within this framework. These prototypes will then be used to evaluate with the applications provided by WP4.

The design of the market protocol factory will be addressed following the approach that has been discussed in section 3.6.

Finally we focus on the integration with other Grid4All system components: (i) trading agents within the autonomic virtual organization framework that has been described in Chapter 1, (ii) semantic based selection of resource markets as described in Chapter 3 and finally (iii) with the component-based overlay services provided by WP1, in particular for the deployment of the market place services.

References

[Alp06] P. Alper, O. Corcho, I. Kotsiopoulos, P. Missier, S. Bechhofer, D. Kuo, and C. Goble. S-OGSA as a Reference Architecture for OntoGrid and for the Semantic Grid. The 3rd GGF Semantic Grid Workshop, GGF16, 2006.

[AUS06] L Ausubel et al., "The clock-proxy auction: A practical combinatorial auction design.", P Crampton, Y Shoham, R Steinberg, eds., Combinatorial auctions, MIT Press, Cambridge, MA, 2006

[Au04] A. AuYoung, B. N. Chun, A. C. Snoeren, and A. Vahdat, "Resource allocation in federated distributed computing infrastructures", in Proceedings of the 1st Workshop on Operating System and Architectural Support for the Ondemand IT InfraStructure, October 2004.

[AWS-ECC] Amazon web services – elastic compute cloud aws.amazon.com

[Baude et al. 06] Baude F., Baduel L., Caromel D., Contes A., Huet F., Morel M. and Quilici R., "Programming, Composing, Deploying for the Grid" in GRID COMPUTING: Software Environments and Tools, Jose C. Cunha and Omer F. Rana (Eds), Springer Verlag, January 2006.

[BAGS02] R. Buyya et al. "Economic models for resource management and scheduling in Grid computing", Concurrency and Computation: practice and experience, (14), 2002

[BAG01] R Buyya, D Abramson, J Giddy, "A case for Economy Grid Architecture for Service Oriented Grid computing" in Proceedings of the 10th Heterogeneous Computing Workshop, HCW 2001

[BAG00] R Buyya, D Abramson, J Giddy, "An economy driven resource management architecture for global computational power grids" in Intl. conference on parallel and distributed processing techniques and applications (PDPTA2000), Las Vegas, USA, 2000

[Bao03] Bao, S. and Wurman, P. R. 2003, "A comparison of two algorithms for multi-unit *k*-double auctions.", In *Proceedings of the 5th international Conference on Electronic Commerce* (Pittsburgh, Pennsylvania, September 30 - October 03, 2003). ICEC '03, vol. 50. ACM Press, New York, NY, 47-52.

[Barm02] A. Barmouta, R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration", 2002 .

[BAR02] Bartolini. C., Preist, C., and Jennings, N. R., Architecting for reuse: A software framework for automated negotiation. In Proceedings of the 3rd International Workshop on Agent-Oriented Software Engineering, Bologna, Italy

[BAV05] Buyya R, Abramson D, Venugopal S. "The Grid economy". Proceedings of the IEEE 2005; 93(3)

[BBG06] C Badica et al., "Implementing rule-based mechanisms for agent-based price negotiations", Proceedings of the ACM Symposium on Applied Computing, 2006

[BCK98] L. Bass, P. Clements, R. Kazman, "Software architecture in Practice", Reading MA: Addison-Wesley Longman, Inc.

[Bella95] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner, "iKP - a family of secure electronic payment protocols", pp. 89-106, 1995.

[Berg06] D. Bergemann and J. Valimaki, "Information in Mechanism Design.", Proceedings of the 9th World Congress of the Econometric Society, 2006.

[Bernhard 01] Bernhard Bauer, Jörg P. Müller, James Odell, "Agent UML: A Formalism for Specifying Multiagent Interaction", *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103, 2001.

[BEN04] M. Benyoucef, R. Keller, "An evaluation of formalisms for negotiations in e-commerce", in Third Intl. Workshop of Distributed Communities on the Web, Quebec City, Canada, June 2000

[Benyoucef & Rinderle 05] Morad Benyoucef, Stefanie Rinderle: A Model-Driven Approach for the Rapid Development of E-Negotiation Systems. EMISA 2005: 80-93

[BGW06] C Badica et al., "Using Rules and R2ML for Modeling Negotiation mechanisms in E-commerce agent systems", in Proceedings of 2nd Intl. conference on Enterprise Application Architecture, Berlin, 2006

[BH01] C Boutilier, H Hoos "Bidding languages for combinatorial auctions", in Proceedings of the 17th International Joint Conference on Artificial Intelligence, 2001

[Brand93] S. Brands, "Untraceable off-line cash in wallet with observers (extended abstract)," in *Advances in Cryptology - CRYPTO '93* (D. R. Stinson, ed.), vol. 773 of *Lecture Notes in Computer Science*, pp. 302-318, Springer-Verlag, 22-26 August 1993.

[BV04] Buyya R, Venugopal S. "The gridbus toolkit for service oriented grid and utility computing: an overview and status report," 1st IEEE Int. Workshop Grid Economics and Business Models, Seoul, 2004

[BY06] R Buyya, C Yeo "A taxonomy of market-based resource management systems for utility-driven cluster computing", Software – practice and experience, 2006, Wiley InterScience

[CA05] R Cavallo et al., "TBBL: A tree-based bidding language for iterative combinatorial exchanges" in Intl. Joint Conferences on Artificial Intelligence, 2005

[CASS] Aaron Cass et al, "Formally defining coordination processes to support contract negotiation", Technical report

[CBF02] Cooper B.F, Garcia-Molina H, "Bidding for storage space in a peer-to-peer data preservation system", Proceedings of the 22nd Int. Conf. on Distributed Computing Systems, Vienna, July 2002, IEEE CSP, 2002

[CHAO02] K-M Chao et. al. "Negotiating agents in a market-oriented grid, Proceedings of the 2nd IEEE/ACM Int. Sym. On cluster computing and the Grid, 2002

[Car02] M. Carman, F. Zini, L. Serafini, K. Stockinger, Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid, submitted for publication. http://citeseer.ist.psu.edu/carman02towards.html. 2002

[Cat05] Catnets Consortium: (Deliverable d3.1: Implementation of additional services for the economic enhanced platforms in grid/p2p platform: Preparation of the concepts and mechanisms for implementation (gmm))

[Cha04] Chase, J., Chun, B., Fu, Y., Schwab, S., Vahdat, A.: (Sharp: An architecture for secure resource peering).2004

[Chau90] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Proceedings of the 8th Annual International Cryptology Conference on Advances in Crytology*, (London, UK), pp. 319-327, Springer-Verlang, 1990.

[Chev06] Y. Chevaleyre, P.E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J.A. Rodriguez-Aguilar, P. Sousa. Issues in Multiagent Resource Allocation, Informatica 30 (2006)

[CHI03] J Chase et al, "Dynamic Virtual Clusters in a Grid Site Manager", In the 12th IEEE Symposium on high performance distributed computing, Seattle WA, June 2003

[Chun05] Chun, B.N., Buonadonna, P., AuYoung, A., Ng, C., Parkes, D.C., Shneidman, J., Snoeren, A.C., Vahdat, A.: Mirage: A microeconomic resource allocation system for sensornet testbeds. In: Proceedings of 2nd IEEEWorkshop on Embedded Networked Sensors (EmNetsII). (2005)

[Chun04] B. Chun, C. Ng, J. Albrecht, D. Parkes, and A. Vahdat. Computational resource exchanges for distributed resource allocation, 2004.

[CHUN04] B.N Chun et al, "Resource allocation in federated distributed computing infrastructures", Proceedings of the 1st workshop on operating system and architectural support for the on demand IT infrastructure, Boston MA, October 2004

[CHYM06] M Carbone, K Honda, N Yoshida, R Milner, "A theoretical basis of communication centred concurrent programming", WCD-Working note, 2006

[CK03] Grid Resource Commercialization, Chris Kenyon, in Grid Resource Management: State of the Art and Research Issues, Editors J.Nabrzyski et al Kluwer, 2003

[CoTy95] B. Cox, J. Tygar, and M. Sirbu, "Netbill security and transaction protocol," pp. 77-88, July 1995.

[CRA98] P Cramton,"Ascending Auctions" in European Economic Review 42, 1998

[CSS06] "Combinatorial Auctions" edited by P. Crampton, Y. Shalom, R. Steinberg, MIT Press, Cambridge, 2006

[CWS89] C.W. Smith, "The social construction of value", University of California Press, 1989

[CZA04] K Czajkowski et al., "The WS-Resource Framework", July 2004

[DES03] L de Silva et al., "Extending agents by transmitting protocols in open systems", in proceedings of 2nd Intl. workshop on challenges in open agent environments, AAMAS03, Melbourne, Australia, 2003

[Delivrable FT 06] "Component Reliability Extensions for Fractal component model " at http://kraken.cs.cas.cz/ft/doc/manual/main-2006-05-16.html#id2451615

[Des04] Z. Despotovic, J.-C. Usunier, K. Aberer: "Towards Peer-to-Peer Double Auctioning", Proceedings of the 37th International Hawaiian Conference on System Sciences, Maui, USA, 2004.

[DR92] Dictionary of economics, Routledge, 1992

[DVS06] de Vries, S et al., "On ascending Vickrey auctions for heterogeneous objects", Journal of economic theory, 2006

[DWP86] The MIT dictionary of modern economics, David W. Pearce editor, MIT Press 1986

[EBAY] www.ebay.com

[EYM03] T. Eymann et. al, "Decentralized resource allocation in application layer networks" in Proceedings of the 3rd Int. Symp. On Cluster Computing and the Grid. Tokyo, Japan, IEEE Computer Society Press: 2003

[EY05] T Eymann et al. "Catallaxy-based Grid markets", in Multiagent and Grid systems", 1(4) December 2005

[FECH05] M Feldman, J Chuang, "Overcoming Free-riding behaviour in peer-to-peer systems", ACM SIGecom Exchanges, 5(4), 2005

[Fer06] Fergus, R., et al., "Removing Camera Shake From A Single Photograph.", ACM Transactions on Graphics, SIGGRAPH 2006 Conference Proceedings, 2006. 25(3): p. 787-794.

[FIPA02] "FIPA Contract Net Interaction Protocol Specification", SC0029H, 2002

[FK02] I. Foster et al., "The physiology of the grid: An Open Grid Services Architecture for Distribued Systems Integration", Globus Project, 2002

[FLZ05] Feldman M, Kevin Lai, Li Zhang, "A price-anticipating resource allocation mechanism for distributed shared clusters", Proceedings of the 6th ACM conference on Electronic Commerce, Vancouver, Canada, 2005

[Fu03] Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat, "Sharp: an architecture for secure resource peering," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 133-148, ACM Press, 2003.

[Garc04] F. D. Garcia and J.-H. Hoepman, "Off-line karma: A decentralized currency for peer-to-peer and grid applications," November 2004.

[Gib73] Alan Gibbard, "Manipulation of voting schemes: A general result", *Econometrica*, 41:587.602, 1973.

[GJJ77] Jerry Green and Jean-JacquesLaffont, "Characterization of satisfactory mechanisms for the revelation of preferences for public goods.", Econometrica, 45:427–438, 1977.

[Grosu04a] Grosu, D., Kant, U.: Mercatus: A toolkit for the simulation of market-based resource allocation protocols in grids. In: SAG. (2004) 176–187

[Gruber 93] Gruber, T.R. “A Translation Approach to PortableOntology Specification.*KnowledgeAcquisition* **5**: 199-220. 1993.

[HAR98] D Harel, M Politi, "Modeling reactive systems with Statecharts: the statemate approach", McGraw-Hill 1998

[HaSt05] D Hausheer, B Stiller, "Peermart: the technology for a distributed auction-based market for peer-to-peer services", in IEEE International Conference on Communications, 2005

[Haus05] D. Hausheer and B. Stiller, "Peermint: Decentralized and secure accounting for peer-to-peer applications.," in *NETWORKING*, pp. 40-52, 2005.

[ICH06] D Irwin, J Chase et al., "Sharing Networked Resources with Brokered Leases", USENIX Technical Conference, June 2006

[IrCh05] D. Irwin, J. Chase, L. Grit, and A. Yumerefendi, "Self-recharging virtual currency," in *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, (New York, NY, USA), pp. 93-98, ACM Press, 2005.

[James Benson al. 2006] Services and Components Based Architectures, White paper from the *Federal Enterprise Architecture Program Management Office*, USA, January 2006.

[Kal03] Kalagnanam, J., Parkes, D.: Auctions, Bidding and Exchange Design. In: Simchi-Levi, Wu, Shen: Supply Chain Analysis in the eBusiness Area, Kluwer Academic Publishers, 2003

[KEL97] F.P Kelly, "Charging and rate control for elastic traffic." European transactions on telecommunications, 1997

[KEN94] Kennedy, G. "Field guide to negotiation", Cambridge, Harvard Business School Press, 1994

[Kev04] Kevin Lai, B.A.H., Fine, L.: Tycoon: A Distributed Market-based Resource Allocation System. Technical Report arXiv:cs.DC/0404013, HP Labs, Palo Alto, CA, USA (2004)

[Kim & Segev] J. B. Kim and A. Segev: A Web Services-Enables Marketplace Architecture for Negotiation Process Management. Int’l Journal on Decision Support Systems. Special Issue on Web Services and Process Management 40(1):71-87 (2005)

[KLAI05] K Lai, "Markets are Dead, long live markets", in ACM SIGecom Exchanges, 5(4), 2005

[KLS04] G Kersten, K.P Law, S Strecker, "A Software platform for multiprotocol e-negotiations", InterNeg Research Paper, 2004

[KN00] G Kersten, S Noronha, "Are all e-commerce negotiations, auctions?", in Proceedings of the 4th Intl. conference on the design of cooperative systems, France, 2000

[KRAU02] Krauter et al, "A taxonomy and survey of Grid resource management systems for distributed computing", Software: practice and experience 2002; 32(2)

[Kri02] Vijay Krishna, "Auction Theory", Academic Press, 2002.

[KSS00] Katrina Stanoevska-Slabeva and Beat F. Schmid. Requirements Analysis for Community Supporting Platforms based on the Media Reference Model. *Electronic Markets*, 10(4):250–257, 2000].

[KWAS05] Kwasnica, A.M, et al, "A new and improved design for multi-objective iterative auctions". Management Science 51(3)

[LAI05] K. Lai et al., "Tycoon: An implementation of a distributed, market-based resource allocation system", Multiagent and Grid Systems – An International Journal 1 (2005)

[Leh02] Lehmann, D., Oćallaghan, L. I., and Shoham, Y. 2002. Truth revelation in approximately efficient combinatorial auctions. *J. ACM* 49, 5 (Sep. 2002), 577-602.

[Lieb04] N. C. Liebau, V. Darlagiannis, A. Mauthe, and R. Steinmetz, "A token-based accounting scheme for p2p-systems," tech. rep., May 2004.

[Li et al. 05] Li Guo, David Robertson and Yun-Heh Chen-Burger, "A Novel Approach for Enacting the Distributed BusinessWorkflows Using BPEL4WS on the Multi-Agent Platform", Proceedings of the 2005 IEEE International Conference on e-Business Engineering (CEBE'05) Beijing, China, October 2005.

[Li 06] Guo Li, "Enacting a Decentralised Workflow Management System on a Multi-agent Platform", PhD thesis from the Artificial Intelligence Applications Institute (AIAI), Informatics , The University of Edinburgh. 2006.

[MCA87] Auctions and Bidding, McAfee and McMillan 1987

[Michael Beisiegel et al. 05] Service component architecture. A Joint Whitepaper by BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, Sybase., Novembre 2005.

[Mica02] S. Micali and R. L. Rivest, "Micropayments revisited," in *CT-RSA*, pp. 149-163, 2002.

[MIL89] Auctions and Bidding: A Primer Milgrom 1989

[MKS03] M Bichler, G Kersten, S Strecker, "Towards a structured design of electronic negotiations", in Group Decision and Negotiation, 2003, 12(4)

[MS01] M. Strobel, "Design of Roles and Protocols for Electronic Negotiations", in Electronic Commerce Research, Special Issue on Market Design 1(3) 2001

[MoP05] P Motty, R Philip, "An efficient multi-unit ascending auction", The Review of Economic Studies, 72(2) April 2005

[Mphq] Mencoder media player, http://www.mplayerhq.hu

[Mur06] Muralidhar V Narumanchi, J.M Vidal, "Algorithms for Distributed Winner Determination in Combinatorial Auctions.", In Agent-Mediated Electronic Commerce, Designing Trading Agents and Mechanisms, Springer 2006

[MuNg01] Y. Mu, K. Q. Nguyen, and V. Varadharajan, "A fair electronic cash scheme," in ISEC '01: Proceedings of the Second International Symposium on Topics in Electronic Commerce, (London, UK), pp. 20-32, Springer-Verlag, 2001.

[Mye81] Robert B Myerson, "Optimal auction design.", Mathematics of Operation Research, 6:58:73, 1981.

[NB06] L Blumrosen, N Nisan, "On the computational power of iterative auctions", in Proceedings of the 6th ACM conference on Electronic Commerce, Canada 2005

[NiRo99] N Nisan, A Ronen, "Algorithmic mechanism design", in Proceedings of the 31st Annual ACM symposium on Theory of Computing, 1999

[NISAN] N Nisan, L Blumrosen, "On the Computational power of iterative auctions", Proceedings of the 6th ACM conference on Electronic Commerce, Canada, 2005

[NIS06] N Nisan, "Bidding languages for combinatorial auctions", In P.Cramton et al., editors, Combinatorial Auctions, MIT Press, 2006

[NPC01] M Nowostawski, M Purvis, S Cranefield, "Modelling and Visualizing Agent Conversations", Intl. Conference on autonomous agents, Quebec, Canada, 2001

[Objectweb]http://fractal.objectweb.org/

[Okam92] T. Okamoto and K. Ohta, "Universal electronic cash," in *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 324-337, Springer-Verlag, 1992.

[OMG99] Object Management Group Negotiation Facility final revised submission

[Parnas 02] David Parnas, The secret history of information hiding. In *Software Pioneers: Contributions To Software Engineering*, M. Broy and E. Denert, Eds. Springer-Verlag New York, New York, NY, 399-409, 2002.

[PAR99] D Parkes, L Ungar, "iBundle: an efficient ascending price bundle auction", in Proceedings of the 1st ACM conference on Electronic Commerce, 1999

[Pat04] P. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. World Wide Web Consortium, February 2004.

[Paurobally 03] S. Paurobally and J. Cunningham. "Achieving common interaction protocols in open agent environments". In Proceedings of the 2nd international workshop on Challenges in Open Agent Environments, Melbourne, Australia, 2003.

[PCr03] P. Crampton, "Electricity Market Design: the good, the bad, and the ugly", in the Hawaii International Conference on System Sciences, January 2003

[Philippe Collet et al. 07] Components and services: A marriage of reason. Technical paper France Telecom, 2007.

[PLA06] Placek et al, "Storage Exchange: a global trading platform for storage services"

[PORT03] D Porter et al., "Combinatorial auction design", Proceedings of the National Academy of Sciences of the United States of America (PNAS)", 100? 2003

[PP03] P Padala et al. "OCEAN: The Open Computation Exchange and arbitration network, a market approach to meta computing", in the proceedings of the Intl. Symposium on Parallel and Distributed Computing, 2003

[PW92] Perry, Wolf, "Foundation for the study of software architecture.", ACM SIGSOFT Software engineering notes 17(4), 1992

[Rege98] O. Regev and N. Nisan, "The popcorn market: an online market for computational resources," in *Proceedings of the first international conference on Information and computation economies*, (New York, NY, USA), pp. 148--157, ACM Press, 1998

[REIN05] M Benyoucef, S Reinderle, "Towards the automation of e-negotiation processes based on Web Services", WISE 2005

[Rene04] R. van Renesse and A. Bozdog, "Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol.", Proc. 3rd Int. Workshop on Peer-To-Peer Systems (IPTPS) 2004.

[Rive04] R. Rivest, "Peppercoin micropayments," in *Financial Cryptography* (A. Juels, ed.), vol. 3110, pp. 2-8, 2004.

[Rive96] R. L. Rivest and A. Shamir, "Payword and micromint: Two simple micropayment schemes," in *Security Protocols Workshop*, pp. 69-87, 1996.

[Rive97] R. L. Rivest, "Electronic lottery tickets as micropayments," in *Financial Cryptography* (R. Hirschfeld, ed.), (Anguilla, British West Indies), pp. 307-314, Springer, 1997.

[RLGW06] D Rolli, "A Descriptive Auction Language", in Electronic Markets, 16(1), Febraury 2006, Routledge

[R2ML] The REWERSE 11 Rule Markup Language

[Rot98] Michael H. Rothkopf, Aleksandar Pekec, and Ronald M. Harstad, "Computationally manageable combinatorial auctions.", Management Science, 44:1131--1147, 1998. http://citeseer.ist.psu.edu/rothkopf98computationally.html

[RUB82] Rubenstein, "Perfect equilibrium in a bargaining model", Econometrica 50(1), 1982

[Sam06] Samee Ullah Khan and I. Ahmad, "Replicating Data Objects in Large-scale Distributed Computing Systems using Extended Vickrey Auction," Intl. J. Computational Intelligence, vol. 3, no. 1, pp. 14-22, 2006.

[SAND02] T Sandholm, W Conen, "Preference elicitation in combinatorial auctions", in Proceedings of the 3rd ACM conference on Electronic Commerce, USA, 2001

[SAND02-1] T Sandholm, "eMediator: A Next Generation Electronic Commerce Server", Computational Intelligence, 18(4), 2002

[San02] Sandholm, T., Suri, S., Gilpin, A., and Levine, D, "Winner Determination in Combinatorial Auction Generalizations.", In Proceedings of the *First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2002

[SCA] Service Component Architecture – Building Systems using a Service Oriented Architecture, Beisiegel

[SCH05] Schnizler et al. "A multiattribute combinatorial exchange for trading grid resources", proceedings of the 12th research symposium on emerging electronic markets (RSEEM), Amsterdam, 2005

[SCH06] Schwind M, "Dynamic resource prices in a combinatorial grid system", The 8th IEEE International conference on e-commerce technology (CEC/EEE'06)

[SEG05] J B Baek, A Segev "A Web Services-enabled market place architecture for negotiation process management", Decision Support Systems 40(1), July 2005

[SETI] http://setiathome.ssl.berkeley.edu/

[ScSc02] Richard E. Schantz and Douglas C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," in Encyclopedia of Software Engineering, John Marciniak and George Telecki, Eds. Wiley & Sons, New York, 2002.

[SHCL96] S.H. Clearwater, "Market-based control: A paradigm for distributed resource allocation.", World Scientific, Singapore, 1996

[SIM04] K.M Sim, "Market-driven agents e-negotiation agents to market-driven g-negotiation agents", Proceedings of the IEEE Int. Conf on e-technology, e-commerce and e-services, HK 2005

[SMI79] A Smith, "An inquiry into the nature and causes of the wealth of nations." W. Straham and T. Cadell, 1779

[SNAP] K Czajkowski et al. "SNAP: A protocol for negotiation of Service Leevel Agreements and coordinated resource management in distributed systems". Job Scheduling Strategies for Parallel Processing: 8th int. workshop Edinburgh 2002

[SNP05] J Shneidman et al. "Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems, USENIX, HotOS'05

[SRT05] S. Ross-Talbot, "Orchestration and choreography: standards, tools and technologies for distributed workflows", NETTAB, 2005

[SW03] M Strobel, C Weinhardt, "The Montreal taxonomy for electronic negotiations", Group Decision and Negotiations, 12(2) Springer Netherlands, 2003

[SunG] Sun Grid Utility http://www.sun.com/service/sungrid/index.jsp

[VDA00] Van der Aalst, "Making work flow: On the application of Petri nets to business process management"

[VDA05] Van der Aalst et al., "Life after BPEL?", in Proceedings of Intl. Workshop on Web Services and Formal Methods (WS-FM), 2005

[Vick61] William Vickrey, "Counterspeculation, Auctions, and Competitive Sealed Tenders", The Journal of *Finance*, Vol. 16, No. 1. (Mar., 1961), pp. 8-37.

[Vish03] V. Vishnumurthy, S. Chandrakumar, and E. Sirer, "Karma : A secure economic framework for peer-to-peer resource sharing," in *Proceedings of the Workshop on the Economics of Peer-to- Peer Systems*, June 2003.

[Vri00] Sven de Vries and Rakesh Vohra. "*Combinatorial Auctions: A Survey*", Kellogg School of Management, Northwestern University. Evanston, IL, Department of Managerial Economics and Decision Sciences, 2000.

[WAL54] L. Walras, "Elements of pure economics", Allen and Unwin (1954)

[WALD92] C.A Waldspurger et al "Spawn: A distributed computational economy". IEEE Transactions on Software Engineering, 18(2), 1992

[W3CCDL] W3C WS-CDL Working Group. Web services choreography description languages version 1.0

[web 1] W3C's website at *www.w3.org/2002/ws/desc/*

[web 2] The ebXML Collaboration Protocol Profile at *www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-cppa*

[web 3] The OASIS UDDI technical specification at *www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec*

[Web 4] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[Szyperski 98] Clemens Szyperski, Component Software, ACM Press/Addison-Wesley, England, 1998

[WOH03] P Wohed, et al., Pattern Based Analysis of BPEL4WS. Technical report, Department of Computer and Systems Sciences Stockholm University/The Royal Institute of Technology, Sweden, Nov. 2003

[WOLSKI01] R Wolski, et al, "Analyzing market-based resource allocation strategies for the computational grid", Intl. Journal of high performance computing applications 2001, (15)3.

[WuWaWe98] P.R Wurman, W.E. Walsh, M.P Wellman, "Flexible double auction for electronic commerce: theory and implementation", Decision Support Systems, Volume 24, 1998

[WWW98] P.R Wurman, M.P Wellman, W.E Walsh, "The Michigan Internet AuctionBot: a configurable auction server for human and software agents", in Intl. conference on autonomous agents, 1998

[WWW01] P.R Wurman, M.P Wellman, W.E Walsh, "A parametrization of the auction design space", Games and Economic Behaviour, Volume 35, 2001

[Yala04] P. Yalagandula and M. Dahlin, "A scalable distributed information management system.", SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pages 379-390, 2004.

[Yang03] B. Yang and H. Garcia-Molina, "Ppay: micropayments for peer-to-peer systems," tech. rep., Stanford University, 2003. http://dbpubs.stanford.edu/pub/2003-31.

[ZaP03] F Zambonelli, H. Parunak, "Towards a paradigm change in computer science and software engineering: a synthesis", The Knowledge Engineering Review, 18(4), Cambridge University Press, 2003

[ZiTi05] Z. Jia, S. Tiange, H. Liansheng, and D. Yiqi, "A new micro-payment protocol based on p2p networks," in *e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference*, October 2005.

[Zu01] E Zurel, N Nisan, "An Efficient Approximate Allocation Algorithm for Combinatorial Auctions", In Proceedings of the ACM Conference on Electronic Commerce, 2001.

Annexes

3.9.1 Market and auction taxonomy

In ***posted price*** models, prices are fixed by the provider. Providers may also make special offers such as discounts for new clients; differentiate prices across peak and off-peak hours. Prices do not vary relative to the current supply and demand but are fixed over a period of time. Commercial utility service offers such the [SunG] and [AWSECC] may be considered as posted price models. The limitation of this model is due to lack of adjustment to fluctuating supply and demand – the behaviour in face of excess demand is not clear.

Bargaining models are employed in bi-lateral negotiations between providers and consumers and do not rely on 3rd parties to mediate the negotiation. Such systems employ strategies that search for joint gains [KEN94]; typically each partner applies concessions until a mutually acceptable utility is reached by alternating offers [RUB82]. Few systems implement bargaining models due to its intrinsic complexity – the success of the model is based on the ability of a consumer to conduct multiple simultaneous negotiations; in general Grid applications require a quantity of resources that exceed the capacity of a single supplier. [SNAP] is a protocol that had been studied in the context of Grids. This protocol addresses the issue of co-allocation, through simultaneous negotiations. In a wide-area network, resources come and go frequently, thus node connectivity and service demands vary frequently [CHAO02]. Efficient resource allocation is aggravated by the volatility of supply and demand [EYM03]. Some projects [CBF02] advocate bartering as being the most appropriate in co-operative peer-to-peer networks that eschew central entities.

Commodity markets use a third party termed market that sets prices based on aggregated supply and demand. These are based on well-founded economic models that determine clearing prices – market equilibrium consists of finding a set of prices and allocation of goods to economic agents such that each maximizes her utility. Markets clear at this price. The "*Tatonnement"* procedure [WAL54] is the precursor to a number of algorithms that have been investigated in the context of markets for computational resources; with an assumption that the demand for each good is equal to its supply. [WOLSKI01] investigates allocation based on commodity markets reasoning that Grids allow applications to treat disparate resources as interchangeable commodities. It determines the price vector of the interrelated commodities at which the excess demand is zero. This approach does not take into account complementarities in resource requests.

Tender model is one of the most used market protocol. Consumers advertise demands, invite providers to submit proposals and consolidate all the proposals to select the most favourable. The Contract Net Protocol [FIPA02] is based on this model is widely used in a large range of electronic negotiations.

An ***Auction*** is the process of trading resources by offering them up for bid and selling the items to the highest bidder. In economic terms it is also a method to determine the value of a resource whose price is unknown. A large amount of research studies bidding strategies and also mechanisms that are incentive compatible. Well known auctions that have been applied in the context of resource management systems are the English, Vickrey, First-price sealed bid, and the Dutch auction [KRAU02]. Combinatorial auctions and double-auctions are also widely studied in the context of computational resources.

Single item auctions: A vast majority of on-line auction sites implement a variant of the English open-outcry auction. Here, the auctioneer begins the auction with a reserve price (lowest acceptable price). Auction continues in rounds with increasing bid prices, until there is no price increase. The item is then sold to the highest bidder. There are many variations on this auction system. Sometimes, the reserve price is not revealed. The auctioneer might do this to prevent rings, groups of bidders who promise not to outbid each other, lowering the final price.

A Vickrey auction is a sealed-bid auction, where bidders submit sealed bids. The highest bidder wins, paying the price of the second highest bid. This gives bidders incentives to bid their true value. When multiple identical units are auctioned, one obvious generalization is to have all bidders pay the amount of the highest non-winning bid. This is known as a uniform-price auction. The uniform-price auction does not, however,

result in bidders bidding their true valuations as they do in a second-price auction unless each bidder only has demand for a single unit.

Single item double auctions: Auctions that simultaneously mediate among multiple buyers and multiple sellers are generically referred to as double auctions. There exist two main institutions for double auctions. (i) The clearing house or call auction, which clears periodically and (ii) the continuous double auction (CDA). The CDA matches buyers and sellers continuously as bids arrive. For homogeneous items, a simple CDA implementation may maintain a queue of bids sorted in increasing order of price and a queue of offers in decreasing order of price. If an incoming bid is greater than the head of the offer queue it is matched to that offer, or otherwise inserted in the bid queue. There are different strategies to set the price for that match. The k-double auction uses a parameter $k \in (0, 1)$ that determines how trades are priced. In that case the transaction price is set at

$$\text{t\_price} = k \times \text{price\_bid} + (1 - k) \times \text{price\_ask}$$

The Mth and (M+1)st double auction computes the Mth and (M+1)st prices where M is the number of sell bids. Such prices may be computed by sorting the bids in descending order and identifying the Mth and (M+1)st elements in the list. The prices between Mth and (M+1)st bids inclusively represent the range of prices for which supply balances demand. [Vick61] demonstrates that the M+1 double auction is incentive compatible for buyers and M double auction is incentive compatible for sellers.

Combinatorial auctions and exchanges: Call markets are appropriate when bids and asks are bundles of heterogeneous items. Winner determination problem in call markets depend on different aspects such as aggregation, resource divisibility and if goods are homogeneous or are heterogeneous. Aggregation can come from the supplier side or from the buyer side. If no aggregation is allowed then each bid can be exactly matched to one ask. Divisible goods can be allocated partially. In the case that the bidder wants the entire good or nothing then its bid is considered indivisible.

For call markets a general linear formulation can be written that captures all the different market structures in terms of aggregation, divisibility and differentiation.

$$\max \sum_{i \text{in} A} \sum_{i \text{in} B} (p_i - p_j) q_i x_{ij}$$

$$s.t \quad \sum_{i \text{ in } B_j} q_i x_{ij} \le q_j \quad \forall j \in A \qquad (1)$$

$$\sum_{j \text{ in } a_{ii}} x_{ij} \le 1 \quad \forall i \in B \qquad (2)$$

$$0 \le x_{ij} \le 1, \forall i, j \qquad (3)$$

(1) Indicates that the quantity allocated to buyers cannot exceed the quantity offered by the seller.

(2) Indicates that no more than the entire good can be allocated.

(3) Specifies that goods are divisible and a portion of them can be allocated.

Combinatorial auction [San02, Zu01, Mur06] is an auction where bidders can bid (sell) entire bundles, that is, combinations of items of goods. This has the advantage of eliminating the risk of a bidder of not being able to obtain complementary (sum of valuations of items is less than the valuation of the entire bundle) items. Market participants express their preferences as bundles of resources that need to be matched. CA is computationally complex. Formally the combinatorial auction problem can be expressed as a special variant of the weighted set packing problem (WSPP) [Vri00].

$$\max \sum_{jinN\ SinM} \sum p_j\, y(S,j)$$

$$s.t \qquad \sum_{j\,in\,N\,i\,in\,S} \sum y(S,j) \le 1 \quad \forall i \in M \qquad (1)$$

$$\sum_{S \subseteq M} y(S,j) \le 1 \quad \forall j \in N \qquad (2)$$

$$y(S,j) = 0,1 \quad \forall S \subseteq M, j \in N \qquad (3)$$

P_j indicates how bidder j values each subset of objects S

(1) Indicates that overlapping sets of goods are never assigned.

(2) Indicates that no bidder receives more than one subset.

(3) Indicates that the subset is indivisible.

The winner determination problem is to label bids as winning or losing so as to maximize the auctioneer's revenue (under the constraint that each item can be allocated to at most one bidder). This NP-complete problem can be solved with dynamic programming techniques that are time intensive. With restrictions on the combinations of bids, winners can be determined in polynomial time – this gives rise to economic inefficiencies since bidders may not be able bid on their preferred combinations. Search algorithms have been investigated to find revenue maximizing allocations – these use bid ordering heuristics, such as ordering bids by their contribution to revenue. A typical bid ordering criteria is the average price per good.

Iterative multi-item auctions

Reasons that motivate the interest in iterative auctions to resolve the resource allocation problem within wide-area distributed computational systems are:

- That the burden of computation is distributed amongst the participants. Based on the feedback on the current prices participants reformulate a bid that maximize their local utility
- It is sufficient for bidders to submit bids only on a small number of bundles within each round.

The approach of iterative auctions is also appealing since it is a straightforward way to trade resources: ask bidders what they would like to buy under certain prices, and increase the prices of over-demanded resources. Bidders at combinatorial auctions are exposed to computational complexity – they need to determine the valuations for all the subset of items that they are interested in, and formulate an optimal bidding strategy given these valuations. Incomplete or incorrect valuations may result in inefficient allocations or reduce the revenues of the auction. Iterative auctions promote the idea that bidders need to only submit the necessary valuations based on the feedback that they receive from the auction.

3.9.2 Rationale for market protocol adaptation

The relations between the concepts of [MKS03] negotiations and auctions need to be clarified before we go further. Traditional auctions are market institutions “with an explicit set of rules determining resource allocation and prices on the basis of bids from the market agents” [MCA87] while the bids indicate the bidder’s willingness-to-pay for the object [MIL89]. These represent a multi-lateral negotiation process based on a fixed-pie assumption [KN00]. In negotiations each party engages in the process to achieve the best possible settlement for them selves and each party seeks to learn the preference structure of the other so as to allow for trade-offs and compromises. The issues (negotiation object) over which negotiations are conducted are composed of multiple criteria and preferences indicate the importance of an objective in comparison with another (for example that reputation is twice as important as the price).

[CWS89] argues that “Real auctions are not exclusively or even primarily exchange processes. They are rather processes for managing the ambiguity and uncertainty of value by establishing social meanings and consensus.” Auctions focus on *determining the value* of objects of unknown value while negotiations are about co-operating to *create value*. Auctions deal with known and well defined objects while negotiations may be about defining these objects and collaborating in order to obtain a common definition. Traditional auctions do not force participants to reveal their utility and preferences – only bids representing the single issue, that is, the price is communicated.

There are two reasons for which we orient ourselves to the negotiation community to address the question of multi-auction support:

- Single-issue auctions are unsatisfactory within Grid resource markets. Recent years have seen efforts to extend traditional auction to address multiple issue auctions and combinatory auctions including iterative auctions that address preference revelations.
- The experience and results generated from the body of research within electronic negotiation platforms are convincing approaches that we can apply within the area of auction mechanisms

In markets, principals and respondents are buyers and sellers that negotiate a price – with or without a mediating process, and goods are allocated at equilibrium prices. The auction protocol repeatedly interacts with the participants and aims to adaptively elicit their preferences [SAND02] so as find an optimal allocation that maximizes the social welfare. These protocols proceed by maintaining temporary prices for the bundles of items and repeatedly querying the bidders as to their preferences between the bundles of resources under the current set of prices and then updates the prices based on the current set of bids.

In practice a number of mechanisms exist – these differentiate themselves in the process of price formation, the feedback generated to the participants, when bids may be generated, and the constraints on the bid itself – the attributes of negotiation (quantity, price) should respect the rules of a given auction design [WWW01], [SPB06], [CSS06].

State of art

Many translations of BPEL into models supporting analysis and verification (process algebras, Petri nets, and finite state machines/statecharts) are currently under investigation.

WS-CDL

The web services technologies framework proposes a specification language called WS***-CDL*** which stands for Web services Choreography Description Languages. [W3CCDL] is an XML-based language that describes peer-to-peer collaborations of parties by defining, from a global point of view, their interaction (in term of ordered message exchange) in order to accomplish a common business goal. ***WS-CDL*** offers possibilities to define the cooperation between a set of abstract roles. In this project we investigate the use of such specification language to define negotiation protocols (roles, interactions, rules termination conditions, etc.). ***WS-CDL*** has a well-defined semantics that may allow an automatic verification of the auction systems properties such as termination securities, etc. thus can avoid ad-hoc negotiation protocols only well defined negotiation protocols can be injected in the e-market. ***WS-CDL*** is now beyond standardisation by the W3C, its symbolic XML can be used by the market to manage or by the actors to share a common understanding of a negotiation protocol.

The WS-CDL describes the protocol by describing the overall interaction between a set of abstract partner (roles). The roles are defined as the observable behaviour of each involved partner type. Commonly the partner observable behaviour is expressed using an abstract version of BPEL. Executing a negotiation protocol consists in composing a set of BPEL Web services in such way to be conform to the WS-CDL protocol specification.

WS-CDL [W3CCDL] was developed by W3C's WS-CDL working group in collaboration with pi-calculus experts as scientific advisors. This describes global information flow and their interaction structures, offering a fully expressive description language for channel based communication.

WS-CDL choreography includes a minimum a set of roles that are defined as some sort of behaviour (i.e. a WSDL description or Abstract BPEL). The roles descriptions represent the notion of a service in the WS-CDL. Relationships between those roles are then defined using channels. Channels allow defining the choreography as a set of typed and unambiguous service connections that enable the various roles to collaborate in order to achieve some common goal. The global view of the roles interaction is defined by using structured composition allowing interactions to be combined into sequences, parallel or conditional activities. The branching condition can be explicit when based on interaction shared data so one can express some data-oriented behaviours rules (of course the conditional behaviours must involve participant who have access to the used data).

The WS-CDL language presents a specific interest for the two main reasons:

- A global description of communication behavior arguably offers conceptual clarity. The natural perspective of a global behavior ensures that the common collaborative observable behavior is not biased towards the view of any one of the participant. This makes it implementation independent.
- Typed nature allow to identify or to specify specific interaction between partners
- The language can support also global behaviors based on exchanged data values this is a central point to complete the negotiation mechanism description by specify constraint (or data oriented rules) on interaction (e.g. a session bid values must be greater than the session open value otherwise a reject message is send back to the participant).

The pi-calculus [CHYM06] experts involved in the WS-CDL working group propose a global description language including, but not limited to, WS-CDL. They have established a well defined operational semantics. They propose formal semantics of a global description language (and by the same to WS-CDL) by an End-Point Project (EPP) to typed pi-calculus. The intuition behind the EPP is that the semantic of a global system behaviors description is given by a sound and complete set of pi-calculus specification called endpoints -- one for each role in the choreography, in the sense that all and only globally described behavior is realized as communications among those endpoints specifications. WS-CDL, being based on the pi-calculus, can be used to show that participant will behave correctly based on advanced static behavioral type checking. From a practical side, the EPP notion also leads as well to significant engineering usage from the global description:

Code generation: Given a global description of a negotiation protocol with full algorithmic details, one can create a (perhaps multi-language) *complete distributed application* by projecting it to each of its endpoints.

Compatibility checking: this point is consequence of the previous Point. Given a WS-CDL protocol description an agent with a specific behavior can check whether its behavior is compatible to play a desired role is the negotiation protocol. The compatibility checking can be performed on the fly (on line) for dynamic instantiation.

Model checking : Both the global description or the an EPP can be translated to one of the pre-cited formal model (FSM or Petri nets see next section) in order to perform model checking, to verify negotiation protocol properties.

Statecharts and Petri nets

A finite state machine (FSM) is a model of behaviour composed of finite states, transitions between those states, and actions. FSM may be represented using a state diagram. The states of the FSM are the states of the negotiation and its input alphabet is the set of messages sent by the participants. The process flow of the negotiation maps into transitions of the FSM. The messages take the negotiation from one state to another. [BEN04] argues that FSM is not sufficient to capture a negotiation process and that it needs to be complemented to answer specific questions concerning the rules and configuration of a given auction protocol such as what is the minimum bid, what is the information that needs to be conveyed to participants; moreover [CASS] argues that FSM is mainly useful to classify negotiations. They describe the changes in

response to events, but fail to describe the order of events. Moreover this notation does not provide an executable form.

Statecharts [HAR98] are gaining more widespread usage since this allows modelling activities as part of a state and to represent concurrent state diagrams. The latter is essential since a negotiation protocol may be considered as the interaction between multiple FSMs. [REIN05] proposes a service oriented negotiation framework where the Statecharts are used to specify protocols. The transitions represent the act of sending and receiving messages – the auctioneer or mediator process waits for an incoming message sent by one of the participants, or by itself sends a message or also that of assignment of a process variable. They use a web service orchestration language based on a reduced subset of BPEL4WS and establish a complete mapping from Statecharts to BPEL4WS. Secondly they also identify commonly used patterns within the models and elaborate corresponding BPEL4WS patterns (such as choice pattern, a pattern consisting of a sequence of message reception and emission etc.). A mapping algorithm then traverses the Statechart graph and maps them to associated BPEL4WS patterns. They have developed a first prototype based on Oracle technology Oracle JDeveloper 10g.

A Petri net also known as place/transition net is one well used mathematical tool to represent distributed systems. A Petri net has place nodes, transition nodes and directed arcs connecting places with transitions. Business process management systems are driven by models of processes and organisations. The process perspective describes the flow of control or the ordering of tasks. [VDA00] promotes the usage of Petri nets to describe work flows. Petri nets allow graphical yet precise specification, offer a number of analysis techniques, and treat states as first class citizens.

[DES03] selects the Petri Net notation to represent local protocols due most of all its ability to represent concurrency in conversations. Petri nets are used to represent the behaviour of each role within the negotiation protocol. This work extends Petri net models to represent external events – messages to send and receive and also internal actions -- useful for modelling local policies and strategies of a participant.

A conversation is a whole Petri Net composed of a set of subnets where at least one role is the privileged initiator. Petri net based approach has nevertheless its limitations – a different net has to be assigned to each agent role, raising questions about how the entire protocol is inferred. [NPC01] presents a layered approach where over and above the net representing the actions of a single role, a *conversation* layer represents the ongoing sequence of messages exchanged between participants.

We need also to cite AUML [BAU99] for Agent UML that extends UML with agent related modelling techniques in particular to represent interactions between agents. It specifies interactions in the form of a UML sequence diagram with extensions. An interaction protocol consists of a number of *lifelines* and messages are depicted as labelled arrows between lifelines. The lifeline indicates flow of time. Even though AUML has advantages due to the visual representation, representation of global views of multi-lateral interactions requires substantial efforts.

3.9.3 Design and Methodology

This section describes the rationale and the technological choice for the design of the Grid4All market place services

Requirements

The Grid poses new challenges in terms of programmability, interoperability, code reuse and efficiency. These challenges mainly arise from the features that are peculiar to Grid, namely heterogeneity and dynamicity as has been discussed in Section 4. Grid4All virtual organizations trade grid resources and services in electronic marketplaces. In this chapter, we focus on the design approach of markets framework for Grid4All.

We first of all need to identify common domain-specific elements of a market process to facilitate the design and implementation of different market behaviours. The design should facilitate reusability, extensibility, and facilitate development of different market mechanisms.

Another requirement for Grid resources market is to support the possibility to externalize services and allow participants to interact with several external infrastructure services (such as discovery services, payment, banking, logging, etc.). Thus, the market framework needs also to ensure dynamicity and interoperability between participants, market services and external ones using standardized and flexible technology.

Finally, in order to enable market participants to publish and discover the traded Grid resources, they need to have at their disposal a formal and semantic description of these resources. This resource description should also support and ensure a common understanding among the trade participants and be able to entitle tools for assisting resource storing and matchmaking (to discover markets, resource availability, etc.).

In the next sub-sections, we will present some proposed solutions that can meet the previous requirements, notably Component Based Software Engineering (CBSE) and Service Oriented Architectures (SOA) for designing the market framework and ontologies for modelling Grid resources.

Component-based approach

A component-based approach meets the previous requirements. The advantages are manifold. Market trading mechanisms can be very complex. Their complexity can be tamed by combining several specialized software components. Component-based approach promotes distributed and autonomous development as well as the reuse of the developed software components for designing new market mechanisms and higher level market services.

Each component has a specified role within the market process, that is, corresponds to different functionality, either market specific or that of a generic infrastructure service. Components encapsulate distinct aspects of the market that may be customized and replaced independently. Due to these advantages, a component-based approach will be used for the design of the Grid4All marketplace. This section, gives first the definition and the main features of components. The possible bridge between components and grid computing is presented after as well as some well known component models.

a. Definition and features

Component-based Software Engineering (CBSE) is concerned with the development of reusable parts, the development of systems from these parts (components), and system maintenance and improvement by means of components replacement or customization. According to [Szyperski 98], a software component is a “unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".

The main features of component-based development are the following:

- Information hiding: Every module hides an important design decision as well as its implementation behind a well-defined interface which does not change when the design decision or the implementation changes [Parnas 02].
- Context independence: Components are easily transferable into different application contexts. As a consequence, components need to be *self-contained* software elements, independent from any other components.
- Implicit invocation: To render components exchangeable, components may ideally not address one another directly, but through an indirection mechanism.

b. Component based approach for Grid Computing

In the context of Grid Computing characterized by highly dynamic, heterogeneous and networked target architectures, it is required that the programming paradigm provides several levels of autonomic management and take care of the functional and non functional features of the architecture building blocks.

Components suitably meet these requirements as they offer the possibility of a reactive control, hierarchical management and contractual specification of the interfaces. Moreover, for the design of a Grid oriented framework, the composing component need to be bound in order to link their provided and required functionalities. To do so a component-based architecture can be described using an ADL (Architecture Description Language) that describes the component system using composition and binding of sub-components. ADL decouples functional program development from the tasks needed to deploy, run and control the components on the component framework.

c. Component models

We present an overview of Fractal [Objectweb], the component model that we have used to specify the market mediator (or auctioneer) process.

- Fractal [Objectweb] is a modular and extensible component model that can be used with various programming languages to design, implement, deploy and reconfigure systems and applications. It enforces separation of concerns, and separation between interfaces and implementation. Components are runtime entities and the interfaces are the only interaction points between components. Dependencies between components are expressed by specifying both the required interfaces and offered interfaces. Fractal provides the notions of a *membrane* and *content*. Membranes permit exercising control over the contents of a component; membranes are implemented as a set of controllers. A Fractal component can export this interfaces to allow introspection of their external features. Four default controllers are specified: binding (bind, unbind client interfaces), attribute (read and write component properties), content (add and remove sub-components), and life-cycle (start, stop interfaces belonging to a component). A second important aspect of Fractal is that the model is recursive. A component may be shared by multiple enclosing components.

  The main goals of the Fractal component model are facilitate development and deployment of complex software systems and provide autonomic self-management capabilities. These goals motivate the main features of the Fractal model: composite components (to have a uniform view of applications at various abstraction levels), shared components (to model resources), introspection capabilities (to monitor a running system), and configuration and reconfiguration capabilities (to deploy and dynamically reconfigure an application).

Service-oriented architecture for market places

Why SOA for Grid resources markets?

Grid resources markets involve different actors which interact in order to achieve trade of these resources. Trade relies on infra-structure services such as information services for dissemination of market situations, discovery services, payments, agreement, etc. That's why, in addition to the component-based architecture for the Grid4All marketplace, a service-oriented approach is also needed. In fact, the use of services for the design of the marketplace helps meeting the following requirements:

- Tolerate and respect the dynamicity negotiating participants relationships between,
- Allow interoperability between the market services and the external ones,
- Use a standardized and flexible integration technology that no organization can afford to ignore if it wants to interact with its partners [SEG05].

In a nutshell, for the development of the Grid4All marketplace, the coexistence of a component-based approach and service oriented architecture (SOA) seems to be suitable. The advantage of combining both approaches is manifold, particularly as web services provide the means for software components to communicate with each other on the web using platform and language independent means.

In the next sub-sections, we present the definition and principals of services, techniques and standards of SOA. Next, we present some state of arts about the convergence between SOA and components. At the end, we introduce the point of view considering markets as business-oriented processes and present some state of arts about business processes modelling languages.

Definition and principals

Service orientation is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

SOA (Service Oriented Architectures) can be regarded as an information systems architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services. These services interoperate based on a formal definition (or contract) that is independent of the underlying platform and programming language. The interface definition hides the implementation of the language-specific

service. SOA-based systems can therefore be independent of development technologies and platforms (such as Java, .NET etc).

The development, maintenance and usage of SOA are conducted according to the following guiding principles:

- Reuse, granularity, modularity, composition, componentization and interoperability,
- Compliance to standards,
- Service identification and categorization, provisioning and delivery, and monitoring and tracking.

SOA techniques and standards

In SOA architectures, a service is a contractually defined behaviour that can be implemented and provided by a component web services as well as J2EE and .NET components. Each service has self-describing interfaces in platform-independent XML documents. Service descriptions consist of the technical parameters, constraints and policies that define the terms to invoke the service. Each service should include a service definition in a standardized format such as W3C's Web Services Description Language (WSDL) [web 1] and ebXML's Collaboration Protocol Profile [web 2].

Communication among consumers and providers or services typically happens in heterogeneous environments -with little or no knowledge about the provider- using formally defined messages via XML Schema (also called XSD [web5]).

Services are maintained in the enterprise by a registry that acts as a directory listing. The advertising and discovery components of SOA can be implemented using a registry/repository or a services directory. The most relevant standard for registry and directory implementations is the Universal Description and Discovery Interface (UDDI) [web 3].

Each SOA service has a quality of service (QoS) associated with it. Some of the key QoS elements are security requirements, such as authentication and authorization, reliable messaging, and policies regarding who can invoke services.

Convergence between SOA and components

CBSE is used to implement the business components; SOA is used to integrate these components by giving an agglomerative view. Within Grid4All, we use component models to design market place services and envisage orchestrating the components using BPEL4WS. We need to understand the integration of CBSE and SOA. Two relevant approaches to integrate the two technologies are SCA and Fractal SCA.

- Service Component Architecture (SCA) [Beisiegel 05] is a set of specifications which describe a model for building applications using component-based model for Service-Oriented Architecture. SCA divides up the steps in building a service-oriented application into two major parts whose result may be a mixture of concepts and mechanisms that are not always homogeneous: (1) The implementation of service components which provide services and consume other services. (2) The assembly of sets of components to build business applications, through the wiring of service references to services
- Fractal WS [Collet 07] is an implementation of services compliant with the Fractal component model in addition to some mechanisms to export components as services: Fractal Toolkit. It is proposed for making compatibility between Fractal components and the web services technology. It transforms interfaces the Fractal components interfaces into web services in order to become accessible through web services protocols. Fractal WS makes also web services accessible inside an assembly of Fractal components using a dedicated proxy component.

From components and services to processes

The Business Process Execution Language for Web Services [Web4] is a notation for specifying business process behaviour based on Web Services. Processes in BPEL4WS export and import functionality by using Web Service interfaces. Business processes can be described in two ways: (1) Executable business processes model the behaviour of a participant in a business interaction. (2) Business protocols use process descriptions that specify the mutually visible message exchange behaviour of each of the parties involved in the protocol, without revealing their internal behaviour. The process descriptions for business protocols are called abstract processes. BPEL4WS is meant to be used to model the behaviour of both executable and abstract processes. It provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL4WS defines an inter-operable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

BPEL4WS fits into the core Web service architecture since it is built on top of XML, XML Schema, WSDL, and UDDI.

By using BPEL4WS, market negotiation mechanism (auction for example) may be specified as a service that can be executed by an orchestration engine. A market negotiation protocol for trading grid resources is modelled as a business process that executes a set of activities performed conforming to a set of rules (negotiation rules). These negotiation processes are asynchronous and may also be long-running [Seg03]; partners are not always connected, and the business negotiation can last a while until reaching a final agreement. Market negotiation processes involve coordinated interactions with other internal and external services. An example of an internal service may be that of *optimization engine* software required to solve the combinatorial auction problem that is expressed as a linear program. An external service may be the information service to which the market process disseminates information when final prices have been determined.

Thus, market negotiation protocols are ideally expressed in terms of executable templates that have some generic APIs. These templates can be implemented as executable business processes using BPEL4WS [Web 4] which is a standard for process modelling based on the Web Services.

Participants may discover the service, and interact with it by sending SOAP messages. From the point of view of participants (buyers, sellers, and other market place service), the BPEL process is seen as a Web Service. Section 8 describes in detail the service operations and messages that are accepted by these operations.

Implementation of market mechanisms using BPEL4WS provides interoperable interfaces to interact with heterogeneous participants. Each process is exposed to the entities taking part in the negotiation process as generic Web Services APIs (bid, query, etc.) that correspond to negotiation messages.

Ontologies for Grid resources description

In the context of Grid4All project, grid resources are heterogeneous and geographically distributed with varying availability and variety of usage for diverse users at different times. In order to enable trade of these resources (computational resources, storage resources) between providers and consumers, they need to be modelled using a formalism that allows their semantic description, for their publishing, storing, and discovery. Besides, the required resource description should support and ensure a common understanding (about resources availability, location, performance, economic characteristics, etc.) among the trade participants as well as the different services (such as resource advertising, retrieving, etc.) contributing to their interoperability.

To enable participants in a trading to share a semantic description of grid resources via a formal language an ontology-based approach seems to be suitable.

Ontology, a definition

An ontology [Gruber 93] is a formal explicit description of concepts in a domain of discourse (concepts), properties of each concept describing various features and attributes of the concept (roles or properties), and restrictions on slots (role restrictions). Ontology is used in order to:

- Share common understanding of the structure of information among people or software agents
- Enable reuse of domain knowledge
- Make domain assumptions explicit
- Separate domain knowledge from the operational knowledge
- Analyze domain knowledge

How ontologies meet our requirements?

The Grid4All resources ontology aims at the description –at an abstract level- of the traded resources (physical resources, computational resources, etc.). This ontology has to sketch out the principal classes and properties of the traded grid resources.

Resources are traded by applying economic models: They are traded as goods, in markets, based on the supply and demand law. An explicit and commonly agreed formal description of resources and their exchange (trading) among information consumers and resource providers is needed in highly distributed and heterogeneous Grid environments.

This shared description should support and ensure a common understanding (communication of knowledge about resources availability and needs) of members (resource consumers and providers) within a market.

4. Semantic discovery – state of art, first architecture

In a market-based environment as envisaged by the Grid4All project, grid resources and services are made available through peer-initiated markets in a distributed manner. The Semantic Information System (SIS) within Grid4All will provide a matching service between peers willing to offer or use resources and services within a Virtual Organization (VO). Grid4all adopts a distributed market model where (resource) consumers and providers negotiate on certain traded entities in auctions initiated by providers, consumers or by third party entities. In order for an auction to take place and given the distributed nature of market creation and management, markets must be discovered by potential participants. A Semantic Information System (SIS) will be a component of the Market-Based Resource Management System (RMS) of Grid4All. The SIS will provide a directory of resources, services and markets, that is, a registry of descriptions of resources and services. By searching though this registry, peers can allocate available resources and services and negotiate through forthcoming or ongoing markets. Markets themselves can be advertised. Provisional provider/consumer software agents and human users will be able to query the SIS registry for available service descriptions that match certain attributes and criteria concerning price, time of availability, quality of a service, etc. The query results shall be ranked according to the capacity of resources, the preferences and intentions of providers and consumers in a selection process.

The SIS will use Semantic Web technologies to facilitate the discovery, matching and selection of services and resources. Semantic descriptions of entities to be discovered shall be stored into the registry. These descriptions will be instances of an ontology that will be developed and used within the SIS. Furthermore, semantic technologies are used for querying and retrieving information from the system. The system will be implemented as a web-based portal.

SIS will implement a *service discovery mechanism* within the Grid4all. It will provide a registry for performing queries in the purpose of discovering available services that fulfill certain criteria imposed by peers within the Grid4All environment. These services are:

- services that expose grid hardware and software resources;
- market services, where grid resources and services are traded;
- services that provide information about other peers that offer or request tradable goods within the Grid4All.

The SIS is not accountable for

- the creation, management and monitoring of markets within the Grid4All environment;
- the creation, management and scheduling of grid services within the Grid4All;
- the invocation of discovered services and matched markets.

The rest of this document aims at specifying the requirements and provide a state of the art of existing approaches related to the Semantic Information System (SIS). In Section 4.1 some background knowledge is provided related to grid resources, grid services and grid economy. In the following Section 4.2, the requirements of the SIS portal are specified, together with a description of available systems for resource and service discovery based on Semantic Web technologies. Section 4.3 describes the requirements and state of the art for the ontology for the description of services and resources, for the semantic matching related to resources and services and for semantic-based service discovery. A description of requirements and related work on the selection service follows in Section 4.5 and the document ends with some conclusions.

4.1 Background Knowledge

4.1.1 Grid Resources

Solving the problem of describing resources/services in a grid, demands considering the dynamic nature of the system (as nodes join or leave the system), of the resources themselves (as these are being allocated to services, being de-allocated, being registered as new nodes appear, or de-registered as nodes leave), specifying resources aggregation (provided by a VO as a whole) and partitioning (for security reasons) at different levels, so as to preserve site autonomy, support policies extensibility, co-allocation of resources and on-line control of the system functions, in conjunction to effectiveness and system efficiency.

Aggregating resources concerns uniting different types of resources to a single whole that can be utilized for supporting one or more services. Aggregation can happen at the level of a single node, at a single VO or at multiple VOs. Pooling of resources concerns gathering a set of available resources, rather than aggregation to a single whole. Partitioning concerns the virtualization of resources (i.e isolation among the jobs and the required resources for fair resource management and secure operation). Autonomy refers to the fact that resources are typically owned and operated (via services) by different organizations, in different administrative domains (a VO is considered to form an administered organization). Therefore it is difficult to expect commonality in an acceptable use policy, preferences on sharing resources etc between users and VOs.

Policy extensibility arises because VOs operate on a range of domains, each with its own requirements. Co-allocation arises because many applications/services have resource requirements that can be satisfied only by using resources simultaneously at different sites. The on-line control problem arises because substantial negotiation can be required to adapt application/service requirements to resource availability, particularly when requirements and resource characteristics change during execution.

Effectiveness concerns the management of resources (status update, monitoring, sampling on their provision and usage, planning, allocation and de-allocation, registering and de-registering, scheduling, discovery, policies adaptation and application requirements etc) so as virtual organizations to provide reliable and trustworthy services. Efficiency concerns the efficiency of resources' management tasks in VOs.

Considering the above issues on resource management in the context of distributed computing systems and virtual organizations, the major issues that we deal with are

- "what types of resources/services have to be described",
- "what are the properties/characteristics of resources/services that have to be represented and be exploited during reasoning and query-answering",
- "what description facilities exist for the types of resources/services we have to consider and what reasoning can be performed for supporting effective and efficient management of resources in these settings".

A resource may be either continuous (e.g. time) or discrete (e.g. a unit of storage space). This "physical" property will typically influence how the resource is being traded within a Grid economy, although this need not be the case. For instance, a continuous resource will typically be regarded as being (infinitely) divisible. Still, in a particular negotiation setting, it may only be possible to buy or sell a certain quantity of such a continuous resource as a whole. Individual units of a discrete resource, however, are always indivisible. In a setting with several continuous resources, a bundle can be represented as a vector of nonnegative reals (or, alternatively, numbers in the interval [0, 1] to denote the proportion of a particular resource owned by the agent receiving the bundle). Bundles of discrete resources can be represented as vectors of non-negative integers. If there is just a single item of each resource in the system, then vectors over the set {0, 1} suffice. A continuous resource (e.g. storage space) may be discredited by dividing it into a number of smaller parts to be traded as indivisible units.

As discussed above, resources may be treated as being either divisible or indivisible. While being either continuous or discrete is a property of resources themselves, the distinction between divisible and indivisible resources is made at the level of the allocation mechanism.

A sharable resource can be allocated to a number of different users at the same time. A resource may be consumable in the sense that the agent holding the resource may use up the resource when performing a particular action. For instance, fuel is consumable. Also, resources may be perishable, in the sense that they may vanish or lose their value when held over an extended period of time.

We call static resources the resources that do not change their properties during a negotiation process. In general, resources cannot be assumed to be static. It is often assumed that they are (that is, those resources are neither consumable nor perishable). The rationale behind this stance is the fact that the negotiation process is not really concerned with the actions agents may undertake outside the process itself. That is, even if a resource is either consumable or perishable, we can often assume that it remains static throughout a particular negotiation process.

Distinguishing between single-unit and multi-unit resources, in a multi-unit setting it is possible to have many resources of the same type and to refer to these resources using the same name. Suppose, for instance, there are a number of bottles of champagne available in the system, but one cannot distinguish between these bottles. In a single-unit setting, on the other hand, every item to be allocated is distinguishable from the other resources and has a unique name. The differentiation between single- and multiunit settings is a matter of representation.

At a sufficiently high level of abstraction, a service allocation problem can be reduced to a resource allocation problem. Indeed, services may be considered resources to which agents assign a negative utility. However, an important characteristic of services as opposed to resources is the fact that services are often coupled with constraints regarding their coherent combination. For instance, a service may require the execution of another service as a precondition.

4.1.2 Grid Economy

Grid computing environments can be based on competitive economic models that provide algorithms/policies and tools for more effective and dynamic resource sharing/allocation [5]. Resources should be able to be traded within e-Markets (by resource providers to resource consumers).

The allocation of resources within Grid computing environments, apart from “conventional style” system-centric scheduling and resource management approaches where a scheduling component decides which jobs can be executed at which site (based on certain static cost functions), can be based on several economic models that provide algorithms/policies and tools for more effective and dynamic resource sharing/allocation. These models, more often, are price-based (as opposed to bartering/exchange-based), where the resources are priced, based on demand, supply, value, and the wealth of economic systems.

Most existing Grid computing environments treat resources as if they all cost the same price even when this is not the case. The end user does not want to pay the highest price but wants to negotiate a particular price based on the demand, value, priority, and available budget. In an economics approach, the scheduling decision is not done statically by a single scheduling entity but directed by the end users requirements. Whereas a conventional cost model often deals with software and hardware costs for running applications, the economic model primarily charges the end user for services that they consume based on the value they derive from it.

Trading based on the demand of users and the available resources is the main driver in the competitive, economic market model: A single user is in competition with other users and resource owners with Grid service providers. In this report we focus on the trading of resource entities in a Grid4All market.

Numerous economic theories including microeconomics and macroeconomics have been proposed in the literature. Some of the commonly used economic models for selling goods and services that can be employed as service price negotiation protocols in Grid computing include [5]:

- **Commodity Market Model:** resource providers specify their service price and charge users according the amount of resource they consume.
- **Posted Price Models**: similar to the commodity market model, except that it advertises special offers in order to attract (new) consumers to establish market share or motivate users to consider using cheaper slots.
- **Bargaining Model:** resource consumers (or their brokers) bargain with providers for lower access price and higher usage duration. Both consumers and providers have their own specific objectives and they negotiate with each other as long as their objectives are met.
- **Tendering/Contract-Net Model**: the most widely used models for service negotiation in a distributed problem solving environment. It is modelled on the contracting mechanism used by businesses to govern the exchange of goods and services. If the provider is unable to provide a satisfactory service or deliver a solution, the Grid resource broker can seek other providers for the service. A task however might be allocated to a less capable provider if a more capable one is busy at allocation time.
- **Auction Model:** supports one-to-many negotiation, between a service/resource provider (seller) and many consumers (buyers), and reduces negotiation to a single value (i.e., price). The auctioneer sets the rules of auction, acceptable for the consumers. Auctions basically use market forces to negotiate a clearing price for the service. Auctions can be conducted as open or closed depending on whether they allow back-and-forth offers and counter offers. The consumer may update the bid and the provider may update the offered sale price. Depending on these parameters, auctions can be classified into four main types:
  - *English auction (first-price open cry)*: all bidders are free to increase their bids exceeding others offers. When none of the bidders are willing to raise the price anymore, the auction ends, and the highest bidder wins the item at the price of his bid.
  - *First-price sealed-bid auction:* each bidder submits one bid without knowing the others' bids. The highest bidder wins the item at the price of his bid.
  - *Vickrey (Second-price sealed-bid) auction:* each bidder submits one bid without knowing the others' bids. The highest bidder wins the item at the price of the second highest bidder.
  - *Dutch Auction:* the auctioneer starts with a high bid/price and continuously lowers the price until one of the bidders takes the item at the current price.

  (Note that for reverse auctions, the roles of provider and consumer is reversed.)
- **Bid-based Proportional Resource Sharing Model**: quite popular in cooperative problem-solving environments like clusters (in single administrative domain). In this model, the percentage of resource share allocated to the user application is proportional to the bid value in comparison to other users' bids. The users are allocated credits or tokens, which they can use for having access to resources. The value of each credit depends on the resource demand and the value that other users place on the resource at the time of usage.
- **Community/Coalition/Bartering Model**: A community of individuals shares each other's resources to create a cooperative computing environment. Those who are contributing their resources to a common pool can get access to that pool. It can involve credits that one can earn by sharing resource, which can them be used when needed.
- **Monopoly/Oligopoly**: a single provider dominates the market and is the single provider of a particular service. Users cannot influence the prices of services (no possibility to negotiate prices) and have to choose the service at the price given by the single provider who monopolized the Grid marketplace. In some cases, a small number of providers (instead of one) dominate the market and set the prices (oligopoly).

4.1.3 Services Description

WSRF (Web Service Resource Framework) [48] specifications, which have recently been adopted by the Grid community, provide a common basis between Grid and Web services. Thus, in a Grid setting, resources

are exposed through Web services which act as their interface. In Web Services community, a Web service is selected after performing a matchmaking process on *services' capabilities*. *Service capabilities* are the actions performed or the information delivered by the web service. Unfortunately, *services matchmaking* is not adequate in cases where grid resources (hereafter resource) need to be discovered. In such cases, *resources matchmaking* mechanisms arises which intent to perform matchmaking on *resources characteristics.* As already explained, r*esource characteristics* are the set of attributes of a resource along with their values. For example, a Hard Disk resource has attribute "capacity" and "model" which have values "50GB" and "SQ-99998", respectively. Therefore, the matchmaking process in a grid setting that adopts the WSRF specifications is characterized as a *resource/service matchmaking* process. Moreover, an indicated difference between Web and Grid services is that the latter ones are potentially transient and stateful, in contrast to web services that are stateless and non-transient.

A typical grid resource/service matchmaking mechanism involves 3 steps:

1. ***Resource/Service Advertisement***. An offer of a grid trade-able resource is made available to the grid via a registry mechanism (resource/service repository). The description of this resource is named resource advertisement.
2. ***Resource/Service Request***. A request for a particular trade-able resource/service is posed to the grid. The request is an expression based on the context of the trade-able resource that will be evaluated in the next step. For example, an expression could be: a Compute Node with more than 2 CPUs of at least 2Gflops each one and a Hard Disk of 80Gbs.
3. ***Resource/Service Matching and Response***. The request is evaluated by the matchmaking mechanism which responds with a list of resources/services that have been matched in some degree (this is relative to the method used for the matchmaking).

Existing approaches for the description and matchmaking of grid resources is achieved on syntactic level based on simple attribute-value pairs with limited flexibility to what can be described. Although, OGSA facilitates interoperability and uniform access to Grid services, OGSA-compliant services are described on syntactic level only.

The Web Services Community aiming to overcome these shortcomings incorporates semantic information to Web services' description resulting in Semantic Web Services technology [45]. Hence, ontologies are exploited for services capabilities understanding in order to bootstrap service matchmaking (selection), composition and monitoring of services. The matchmaking is performed on inputs and outputs parameters of the advertised and requested descriptions of web services based on the available ontologies. There are three main approaches that aim to incorporate semantics in Web services.

1. *OWL-S*. It is an agent oriented approach to SWS that has been evolved from DAML-S [11]. It provides an upper level ontology (namely OWL-S) for describing the capabilities of Web services in unambiguous, semantically enriched and computer interpretable way.
2. *Web Services Modeling Framework* [46]. It provides the WSMO [47] conceptual formal model for the description and annotation of all the relevant aspects of services. This approach introduces the notion of mediation between the components realized in an e-commerce application. The approach is consistent with the principle of maximal decoupling and scalable mediation.
3. *WSDL-S*. This approach is the evolution of the METEOR-S [28] project aiming at allowing semantics descriptions of Web services on the ground level. Hence, this approach does not separate service grounding from service profile. The WSDL is augmented with semantics resulting in WSDL-S language. This is performed by linking the WSDL's elements to concepts of externally defined ontologies.

The last one is a bottom-up approach, providing semantic at the lower level of a service, in contrast to the first two one which characterized as top-down approaches. In the literature, only the presented approach implements the bottom-up incorporation of semantics in Web services.

In the following lines we place important questions regarding the matchmaking process:

i. Does the resource request and offer use a common-shared ontology for their description? If this is not the case then what type of techniques and methods are used to overcome this type of heterogeneity?
ii. Is the matchmaking process performed only on IOEP service's attributes? If this is not the case what other information is used. For example, quality of services information or usage polices can be exploited in matchmaking.
iii. How ontology is used in the matchmaking process? For example, is it used
    a. as common vocabulary for describing resource requests and offers,
    b. for inference and what type of inference,
    c. queries (e.g. SPARQL) expressed on ontology,
    d. in any other way
iv. What degree of matching is supported? For example, exact match (true or false on matching) or confidence match (range of match) etc.
v. Is uncertainty on service's attribute values handled? For example, a service request may not specify values on some attributes.
vi. Does the matchmaking process support any matching preferences? For example, restriction on the number of result matches etc.

4.2 Semantic Information System Requirements and related work

This section describes the functional and the non functional requirements of the Semantic Information System. It describes the relationship of the SIS with other components in the Grid4All, the types of entities (actors) that interact with the system, the functionality that the system exposes to its users and the characteristics of this functionality. The description focuses on the interaction of the SIS with its users rather than the description of the information that is exchanged during SIS-user interaction. This information is described in detail in Section 4.3.4 where the ontology and the matchmaking process requirements are presented.

4.2.1 Semantic Information System Requirements

Relationship with the Resource Management System

Within the Grid4All, a Market-based Resource Management System is responsible for the creation and management of markets in a distributed manner. Potential users of grid services and resources, that is, *resource consumers*, must discover the *providers (owners)* of intended services and resources. Conversely, consumers must discover providers that offer resources and services they need. In a distributed market environment, consumers and providers must discover markets where appropriate resources/services are being traded. The Resource Management System depends on the Semantic Information System for the discovery, matching and selection of traded resources and services.

A market in the RMS has the following states: First the market is created, or initiated by, either a provider who wants to trade an entity in a forward market, or a consumer that wants to buy an entity. In the latter case it is called a reverse market. Then, an offer is specified by the initiator of the market. Next, the market together with the corresponding offer is advertised in the SIS. The offer is discovered by potential participants in the market by using the SIS. The market is activated by the initiator and closed after the completion of the auction process. The main role of the SIS is the advertisement and the discovery of markets. There are three kinds of markets:

- Provider-initiated markets
- Consumer-initiated markets
- Third party-initiated markets.

The SIS is used for the discovery of markets and services that trade or expose grid resources and services with desirable characteristics. Service discovery and matching between requests and offers is performed based on static information, that is, information provided before the actual trading processes are activated in

the context of the Resource Management System. Thus, the SIS shall not handle updated/dynamic information on ongoing markets.

Updated information concerning market/auction status will be stored in the Market Information Service (MIS) component of the Resource Management System. The MIS is a service in charge of aggregating information of the markets. Another function of the MIS is to deal with dynamic information such as resource availability or price changes. Thus, the SIS must provide an appropriate API for the interoperation with the Resource Management System to facilitate the synchronization of information between SIS and RMS. This API will be specified in later stages of the SIS development.

The market factory component of the resource management system creates trading sessions (markets) and advertises the opened sessions at the SIS registry. This service provides a set of administrator interfaces to allow the creation of market/trade sessions. Market sessions may be initiated by registered/authorized consumers, suppliers, or by 3rd party mediating services. Auctions created by the Market factory can be advertised into the SIS by the factory. That is, the market factory can insert descriptions about created markets into the SIS. The SIS must provide an API in order to facilitate the advertisement of market descriptions from the market factory.

SIS Actors

SIS actors are entities external to the SIS which interact with the system. Such entities are human and software agents, or external components and services. The SIS is expected to interact with the following actors:

- **Provider**: The owner of a specific resource or service. Providers trade their services or resources though a market auction initiated either by them or by another entity. A provider, also named ‘Seller’ or ‘Supplier’, can be both a human user and an agent acting in the context of the Resource Management System.
- **Consumer**: An entity which intends to acquire a specific service or resource with predefined technical characteristics, capacity and quality of service measures. A consumer, also named ‘Buyer’, can be a human or a software agent acting in the context of the Resource Management System.
- **Market factory**: The market factory is an RMS service described above. It interacts with the SIS for advertising/registering markets initiated by providers and consumers.
- **MIS**: The MIS is an RMS service described above.

User Interfaces

The SIS shall be implemented as a portal accessible though the WWW by a web browser. The user interface will be composed of web-based forms. The users are not intended to directly manipulate ontologies according to which market specific information concerning tradable resources and services is going to be specified. That is, formal ontology descriptions must be transparent to the users of the system. This will be achieved by providing templates or wizard-driven interfaces that will be automatically created (on-the-fly) from the ontology schema descriptions. This mechanism will also support the dynamic evolution of the ontology without being necessary to concern about the evolution of the SIS interface itself. An additional model is needed for providing information concerning the position or presentation sequence of the formal descriptions in the SIS GUI.

Software Interfaces

The SIS will provide both a user interface and a corresponding API, exported as a set of web services. The purpose for a web service implementation is:

- to provide access to the system features by agents acting in the context of the Market-based RMS, i.e. consumers and providers,
- to facilitate interoperation with Market factory and MIS,
- to facilitate the automation of system testing and benchmarking.

System Features

The functionality of the SIS is described as a set of features provided by the system. The major features provided by the SIS are:

- publishing or advertising of market-related request or offer information, as well as information about traded resources and services,
- editing/deleting (canceling) requests or offers 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.

System features are described in the following section in terms of user interaction, input and output, prerequisites (pre-conditions) and results (post-conditions).

Publishing/advertisement

The purpose of this feature is the insertion into the SIS registry of a specific offer or request by a provider or consumer. These descriptions contain information about the entities that are traded by the associated markets, that is, resources and services, as well as information about the related markets, the participants, providers and prospective consumers of resources and services. Descriptions will be instances of the ontology schema that will be developed in the context of the SIS. The description will be in the form of object-property-value triples. As mentioned above, publishing will be supported by a web-based user interface for human users. No authoring of formal descriptions of the input information will be required from the human users in order to create and submit an ontology instance. A corresponding API will be available as a web service for the automatic publishing of descriptions by agents and the Market factory service.

Advertisement will be supported in different ways for consumers and providers.

Providers will advertise into the SIS registry descriptions of markets that will sell resources/services. These are descriptions of forward markets in which providers will sell resources/services to potential costumers. As already pointed, advertised information from a provider is called an *offer*. The exact form of data that constitute an advertisement is described in sections 4.3.2 and 4.3.3. Providers shall be able to advertise a list of XOR offerings, that is, alternative service/resource configurations at different prices, availability times etc.

Consumers will advertise into the SIS descriptions of markets in which they will call for resources/services of given characteristics. A market advertisement from a consumer refers to a reverse market which is initiated by a request or RFQ (Request for Quotation). Reverse markets are initiated by Consumers in the context of the Market-based Resource Management System. The information must be an instance of the Ontology description for reverse auction markets.

A market is to be advertised directly by its initiator, Provider or Consumer, or through the Market factory which instantiates this market.

Offers and requests are *market orders* and contain the following information:

- *Market related information:* The description of a *market* where the resource/service is going to be traded: This includes location of market, start and closing time of market.
- *Traded resource/service related information*: The description of the technical characteristics of the traded services or resources in terms of *capacity, quality of service, time of availability, etc*.
- *Offer/request related information*: The description of pricing policy (type of related market auction), initial price auction price (minimum price for a forward auction and maximum for a reverse auction).
- *Contact information:* Information about the provider or consumer.

System Use Case description

The following steps describe the interaction between a user, either provider or consumer, and the SIS portal user interface during the advertisement of a market order:

- The SIS shall check whether a user has successfully logged into the web-based system.
- The user selects whether he/she wants to insert a market request or a market offer.

- The system displays the appropriate forms depending on whether the advertisement is related to an offer or to a request.
- The user inserts information related to the market order: Information about market orders contain locality, time of availability, type and quality of resources, type of auction, initial price of the auction. In the case that the user is a consumer, he or she may not provide exact information but desirable constraints about the resources/services that he wants to buy. Given that the advertiser is authenticated into the system, information about the initiator of the market is automatically added into the advertisement. Price units are inserted in a defined form of virtual currency.
- The user inserts a description of the resources and services that are traded in a particular market. This description contains information on capacity, QoS, configuration of the traded resources and characteristics of services. The user interface shall provide tools for the adequate description of complex resource configurations and aggregations, as well as descriptions of services as they are described in sections 4.3.2 and 4.3.3. Alternatively, market-related information can be automatically provided by the Market factory.
- The user submits the offer or request.
- The system translates the information into appropriate SIS ontology classes and instances. The system validates the inserted data. In case of inconsistent data it returns an error message, else, a confirmation message is presented to the user.
- The system notifies subscribed users interested in the particular type of advertisement. The subscription mechanism is described later in this document.

Input and output

The input data format for this feature is a description of a *Market Order*. A *Market Order* is an *Offer* made by a *Provider* or a *Request* made by a *Consumer*.

There is no output data related to this feature. The SIS returns an acknowledgement in case of a successful *Market description* advertisement. In case of an unsuccessful advertisement the system throws a software exception and displays appropriate error messages in the user interface.

Restrictions

Precondition

In order to be advertised, a market must have been created by its initiator in the Resource Management System.

Postcondition

After that scenario, a market description for the specified resources exists in the SIS registry. The query is stored in the log of the SIS.

Querying of resources/services

In the context of Grid4All, service/resource providers try to find appropriate consumers and, respectively, consumers try to find appropriate providers.

The purpose of query feature of the SIS is to provide to prospective providers/consumers an ordered list of available *markets,* already published into the system, that fulfill certain criteria concerning their own characteristics, as well as the characteristics of the traded goods: performance and QoS characteristics of a service, or the configuration of resources, pricing, market and location of markets, resources and services criteria. Querying answering is based on *matching* and *selection*. Matching is the identification of a set of semantic descriptions of markets that fulfill the criteria imposed by the query. The requirements of the matching sub-feature are presented in Section 4.3.3. Selection refers to the ordering of the advertised matched markets according to certain characteristics including the capacity of resources, preferences and intentions of providers and consumers. The requirements of the selection sub-feature are presented in Section 4.4.4.

A provider or a consumer (human or software) agent submits query data in the form of query attribute values. A particular query constitutes a class or instance of the SIS ontology schema. This instance is matched over the ontology instances that are already advertised in the SIS repository. Queries are inserted by users using an appropriate web-based user interface and the users do not need to know any ontology specific query language. The result of a query is an ordered list of market descriptions that the searcher (provider or consumer) can exploit in order to acquire the corresponding resource/service. The result of a query can also be a list of providers or consumers.

Queries in SIS will be *authenticated*. Only registered users who have successfully logged into the system can use the query functionality.

Providers and consumers can also subscribe to the SIS by submitting a query with the intended traded entity information. In this case, whenever a new market advertisement takes place, if this advertisement matches with the query data, the SIS notifies the interested provider or consumer. This notification service can take place synchronously, on the time of the advertisement submission, or asynchronously, for example on a periodic basis.

Consumers perform queries in order to obtain information about available resources and services. There are various scenarios for consumer search:

A consumer wants to apply a bid to a market initiated by a provider. The consumer provides a set of attribute values that describe the intended configuration and/or characteristics of the services/resources that she needs. These attributes are related to price intervals (maximum and minimum), quantity and quality of service, service capacity, local deadlines, and time intervals. The exact information related to requests and offers is specified in Section 4.3.3. As an example of simple case, a consumer may need a resource with the following characteristics:

- A compute node with CPU resources of an Intel32 compatible processor with at least 1 giga memory
- the consumer needs the resources between 7:00 and 17:00 hours Athens local time
- the consumer needs the Compute nodes for 2 hours
- the consumer may not want to specify the real total number of CPUs, but only that a minimum and maximum number of CPUs are required.

The services/resources are going to be traded in a provider (or third-party) initiated market. Attributes of such a market may also be specified in the search query. The system returns a list of advertised markets. The consumer selects one or more market services from the list and submits a bid to one of these markets. The consumer may also provide an XOR query, in which *one request in a set of alternatives should be allocated.*

A consumer may want to start a *private reverse market*. In this case the consumer wants to discover a set of providers according to certain constraints: As another example, a consumer may ask for a minimum of 4 compute nodes and a maximum of 6 compute nodes where every compute node should be allocated at the same starting time and for the same interval duration. The consumer will specify the earliest starting time and the latest ending time as well. The query will result in a set of contact information about matched providers. The consumer will use this contact information in order to invite a selected subset of the matched providers to participate in the reverse market initiated by the consumer.

Resource/service providers will perform queries in order to discover consumer-initiated markets (reverse auction markets). An example scenario is the following:

A Provider wants to discover all Requests for Quotations that try to acquire 10M instructions in no more than three Grid Compute Node Resources from the same provider with a 32-bit processor with ram no less than 1GB and speed at least 1.0 GHz starting all at the same time and starting no later than 10 minutes past now in Barcelona.

User subscription/un-subscription. An agent, provider or consumer, may want to be notified when market descriptions that match query criteria are advertised into the SIS. On this purpose, the provider submits the query information to the SIS. As a result, the SIS returns a list of matching market order descriptions. The user then may *subscribe* to the SIS in order to be notified when market orders matching the above criteria

are advertised into the SIS. The user can cancel the subscription at a later time. Both subscription and un-subscription can be considered as extensions of the query feature/use case.

Use Case description

The following steps describe the querying functionality of the SIS.

- The SIS shall check whether a user has successfully logged into the system.
- The user specifies whether he/she wants to perform a query as a provider or as a consumer of a tradable entity.
- The system displays the appropriate forms depending on the type of the user.
- The user inserts information related to the query. The query can be based upon a combination of conditions that specify the desirable features of services, resources and markets, including locality, availability time intervals, type and quality of resources, type of auction, initial price of the auction, type of market and corresponding auction, that is, is the maximum price for a reverse auction and a minimum price for a forward auction. Price units are inserted in a defined form of virtual currency.
- The user inserts a description of the desirable resource and service characteristics that will be traded in a particular market. As in the case of advertisement, if the user is a consumer, he or she may not provide exact information but only desirable constraints about the resources/services that he wants to buy. The user interface shall provide tools for the adequate insertion of complex resource and service configurations, that is, aggregations and compositions of resources and services as they are described in Section 4.3.2.
- If the user is a consumer, he or she can impose certain constraints on the resource configuration and market related characteristics.
- The user submits the query.
- The system translates the information into an appropriate SIS ontology query. The system validates the inserted data. In case of inconsistent data it returns an error message, else, a confirmation message is presented to the user.
- As mentioned before, the user may determine whether he/she wants to be notified when matching advertisements are registered into the SIS.

Input and output

Query information of users is the input for this feature. The output is either a list of market descriptions either a list of market initiator contact information.

Restrictions

Preconditions

No preconditions exist for submitting a query. If no matches are found, the returned list is empty.

Postconditions

The query is logged into the system.

Browse/list available services/resources

The SIS will provide a directory index i.e. a visual representation of a list of available service/resource and market descriptions for browsing. The descriptions shall be categorized according to the type of the described entity, hardware or software resource, application domain, etc, according to their descriptions. An indicative hierarchy of categories is the following

- Resource
    - CPU Resource
    - Storage Resource
    - Compute Node
    - Cluster
- Application
    - Multimedia

- Scientific computation

Alternative browsing by market characteristics shall be available. The service/resource consumers/providers will select their preferred categories and be navigated through the service catalog through a web-based user interface.

Authentication/ authorization

The SIS portal shall provide a login mechanism for human users. For software agents, the access to SIS services API shall be controlled by using a web service security mechanism such as HTTP-based security, SSL or WS-Security.

User Management

The SIS portal shall provide user management functionality. Privileged portal users (administrators) shall be able to

- Add a user to the system
- Modify user information
- Delete user information.

The users themselves should be able to modify their profile into the portal. Registered users can be both resource/ service providers and consumers.

Logging

The SIS shall provide a logging feature. Advertisements and queries shall be recorded into the system. Logging shall be used for system evaluation and monitoring purposes.

4.2.2 State of the art of Semantic Information Systems for the grid

In the following paragraphs available systems and software frameworks that provide matching of resources and services in grid environments are described. The provided descriptions focus on the requirements and features provided by these systems and frameworks as well as on the use of semantic technologies in order to meet these requirements.

In [15] a system for matching resources to requests (applications) of *users and agents* in a grid environment is presented. The OMMS (Ontology-based Matchmaker System) adopts an extensible approach for performing Grid resource selection that uses separate ontologies to declaratively describe resources and job requests. Thus, the matching between job requests and available resources is performed in a semantic rather than in syntactical level. Matched resources are ranked according to user preference using a ranking function, which is an arithmetic expression expressed in terms of resource properties. The OMMS service dynamically discovers and updates information about available resources utilizing two mechanisms: The subscription/notification mechanism which is provided by the Grid infrastructure (Globus Toolkit) and a polling mechanism of available resources with incremental description updates.

In [26] a matchmaking framework for semantic service discovery in Grid environments and a portal system which implements this framework are defined. The aim of the framework is to provide matches between applications and available services. Semantic matchmaking is based on reasoning based on structured information about available services and applications, rather than mere syntactic matching based on service/application attribute name comparison. The framework fulfils a number of requirements such as high degree of flexibility and expressiveness, support for subsumption (reasoning), support for datatypes, efficiency, compliance with existing Grid technologies and capability of lookup and invocation of matched services. The proposed matchmaking is based on a shared ontology and is defined in three stages: Context selection, where the appropriate context ontology is chosen, semantic selection, where requests are matched to services according to the metadata descriptions of services, and registry selection, where services are looked up in a UDDI registry. Service descriptions are defined in the DAML+OIL ontology language. A service discovery portal implements the framework, supporting the advertising, viewing and searching of services. The framework provides a similarity metric for avoiding the exploitation of the

matching mechanism by too generic advertisements or requests that attempt to maximize the likelihood of matching and for facilitating ranking of selected services.

In [31] a solution for web service advertisement and matching based on a semantic description of web services is described. This description consists of a combination of DAML-S and UDDI. The requirements for this matching services are flexibility, minimization of false positive and negative matches, and efficiency. DAML-S descriptions are used for formalizing the functionality of the advertised web services using service profiles, while UDDI is used for syntactic keyword-based matching. An implementation, of a DAML-S/UDDI Matchmaker is provided. An algorithm for matching of DAML-S descriptions of services is proposed and implemented.

A number of systems have been developed for the discovery of domain-specific Grid resources and services based on semantic descriptions for specific application domains such as Computer-supported Collaborative Learning [43], Bioinformatics [44] and Meterology [35]. These systems are based on Semantic Web formalisms for the annotation of resources and services such as OWL-S and DAML+OIL. The ontologies on which they are based are domain specific, that is, they define concepts that have a specific meaning in the specific context of the application, and thus support refined queries, meaningful for experts in the specific field.

In [43] a system for the discovery of learning services in the context of Computer Supported Collaborative Learning (CSCL) is described. This discovery service is a component of a grid-based CSCL system named Gridcole. A semantic approach is adopted in order to overcome the shortcomings of keyword-based matching approaches such as UDDI. Users of the Gridcole system are practitioners in university education, which build CSCL scenarios by setting up *educational activities*. These activities are supported by specific collaborative learning tools which are implemented as grid services. Examples of such tools are e-mail, chat and news for communication, collaborative editors and argumentation tools and shared repositories for knowledge sharing. Users query the discovery service in order to locate such tools. The users may, for example, search for a chat, a simulator for a specific course, a tool for a collaborative editing of an MS Word document by a group of four members, a tool for document exchange.

In order to facilitate such queries, an ontology which incorporates concepts from the specific field of collaborative tools to enable their discovery by educators is developed. This can be used to annotate any type of CSCL tool, implemented as a grid service, a web service, or a collection of Java classes.

The ontology is defined using OWL. Ontology consistency is checked with the Racer reasoner. The descriptions of available services are stored as ontology instances in a central repository. Queries are performed used the Racer Query Language (RQL). It is envisaged that the system should provide an appropriate user interface so as that no knowledge of RQL is needed for forming queries without limit the expressivity of RQL and OWL.

In [35] a semantic framework for a grid meteorology computing environment is described. The framework supports the discovery, selection and workflow composition of meteorological services. In this framework semantic matchmaking is based on an OWL inference mechanism based on the JESS [17] rule-based engine. During semantic matchmaking, submitted requests are matched to advertised service descriptions. Advertised services are described by a combination of UDDI and OWL. The ontology used, which is described in OWL, refers to the domain of meteorology. The ranking of matched services is based on a quality of service model.The framework also supports service composition, that is, the combination of several services in order to provide some functionality required by the user. Composition is supported in design time, when a user can define a sequence of services that will be executed upon demand, or in run-time, when services that match certain criteria are automatically combined, transparently from the point of view of the user. Combined services are discovered using the semantic matching facilities of the framework. Submitted request descriptions conform to the WSDL-S specification.

4.3 Grid4All Ontologies and matchmaking for the Grid: Requirements and Related Work

4.3.1 Introduction

Many types of Grid systems aim to support remote and/or concurrent use of geographically distributed and heterogeneous resources. Such systems usually use a resource management system for assembling and using collections of computational and/or storage resources on an as-needed basis. In the context of the Grid4All project, where the Grid is seen as a ubiquitous utility for users and small organizations, such Grids are considered to form one or more virtual organizations (VO). A VO is considered to form a “wrapper” for resources on which services are deployed and which will be exposed to the users.

In the view of a Grid4All VO, a resource is anything the members offer (sell) to the VO that other members can request (buy). A service on the other hand is how the VO provides access to these resources brought in by its members, and how the VO provides access to their discovery, management, scheduling and monitoring. In the Grid4All view, resources are traded by applying economic models: They are traded as goods, in markets, based on the supply and demand law.

An explicit and commonly agreed formal description of resources and their exchange (trading) among information consumers and resource providers is needed in highly distributed and heterogeneous Grid environments. This shared description should support and ensure a common understanding (communication of knowledge about resources availability and needs) of members (resource consumers and providers) within a VO, but also between different Grid infrastructures by contributing to their interoperability.

In the following paragraphs we first outline the requirements for service/resource ontologies in the frame of Grid4All project. We then provide the requirements of the technological aspects concerning the implementation of such an ontology, as well as the matchmaking of resources request/offers at the semantic level. Next we provide an extended review of state-of-the-art related approaches on a) resource ontologies and related aspects, b) grid economy and c) semantic matchmaking, and also technologies that support the development, reasoning, querying and storing of ontologies. In the following subsections we present available QoS and policy related to web services ontologies. This section ends with state of the art on resources/services matchmaking.

4.3.2 Requirements for a Grid4All Services Ontology

Recently, Open Grid Service Architecture (OGSA) [29] has adopted the service-oriented architecture based on Web services concepts. The result of this adoption is the WS Resource Framework (WSRF) which re-factors and evolves the Open Grid Service Infrastructure (OGSI) [14] and constitutes a common base between Grid and Web services communities. OGSI introduced the idea of stateful Web Services and defines the approaches for creating, naming, and managing lifetime of service instances. This notion, as well as all the functional capabilities presented in OGSI, has been retained by WSRF. In a Grid setting, Web services may act as interfaces for the exposition of grid resources. Moreover, resources are not restricted to application software as is the case of a regular Web Services. WSRF is a suite of specifications that defines Web Services interfaces for creating/destroying resources, managing their lifecycle, inspecting and disseminating their properties.

Although, OGSA facilitates interoperability and uniform access to Grid services, OGSA-compliant services are described on syntactic level only. Also, this characteristic has great impact on the performance of the service discovery mechanism. To successfully find a particular Grid Service, one must know the exact names of the service, its operations and data types.

In the Grid4all context a service should be able to expose the following types of resources:

1. *Compute Nodes* and *aggregation of Compute Nodes* (namely Clusters),
2. *Software*, *applications* that are able to run in grid environment,
3. *Markets*, where grid resources and services are traded
4. *Services* that provide information about other peers that offer or request tradable goods.

An Application is a program that can be moved to and executed by any Compute Node (CN) of the grid. On the other hand, a Service is heavily coupled with a CN and its resources (software), which in most of the times are not visible to the grid. For example, an “avi2mpeg” application can run in any CN but a “book a room” service cannot since it needs resources that cannot be found in other CNs. A Market is a specialization of Service since it aims to provide a service mechanism for trading resource/services.

Also, Grid4All service description must conform to:

a) *Data Semantics*. The meaning of the data should be semantically specified. For example, the input and output data of the service’s operations should be semantically described.
b) *Functional Semantics*. Services operations, i.e. the transformation of input data to output data, should be semantically specified.
c) *Execution Semantics*. Preconditions and effects of services should be semantically specified. For example, a service that charges consumer’s credit card should be stated at the precondition as well as that the verification of the consumer that will have erased the result data after reading it should be stated at the post-conditions constraints.
d) *Quality of Services (QoS) and their Semantics.* Services should be associated with QoS aspects and their semantics. Availability and level of trust are some examples of QoS metrics.
e) *Authorizations Policies and their Semantics.* Services should be able to incorporate authorization policy aspects. For example, what group of users can access a specific resource or a group of resource?
f) *Usage Policies and their Semantics*. Services should be able to incorporate usage policy aspects. For example, what time is the resource available for a specific group of users?
g) *Easy extensibility of services’ characteristics*. Characteristics and concepts that used in the description of a service should be extensible.
h) *Support of describing Resources’ states and their Semantics*. Services should be able to provide access to the semantic descriptions of the states of a resource.
i) *WSRF specification grounding.* The description of service should conform to WSRF specifications and W3C standards.

In addition, several thematic sub-ontologies (described in other sections) should be developed for or adopted to Grid4all requirements in order to support Grid4all services’ requirements semantics.

Service Matchmaking

Resource discovery (and as a consequence) service discovery is an important issue in the Grid environment, since it facilitates the search of a service that meets some specific requirements and its successful exploitation.

A service matchmaking mechanism should support:

1. the asymmetric matchmaking of offers and requests.
2. the inference of s*ubsumption relations between descriptions*.
3. the use of c*ustom data types for values.*
4. th*e exploitation of QoS aspects.*
5. the use of Polices’ aspects.
6. the discovery of services that expose particular type of trade-able resource.
7. the discovery of services that expose arbitrary number of trade-able resources.

8. the discovery of services that exposes arbitrary configuration of trade-able resources.
9. the ability to describe matching preferences. Both request and resource can specify their preferences when multiple matches are found.
10. the ability to answer complex resource matching. It must support partial specifications and/or creation of requests/offers that synthesize many concepts.
11. unify advertisement view of services and resources (web services and web services that expose resources)
12. representation and exploitation of resources' state
13. Association between a stateful resource and a stateless service (WS_Resource).
14. Manage life-time and properties of these associations
15. Realize the instantiation of WS-Resources
16. The integrity checking of results

4.3.3 Requirements for a Grid4All Resources Ontology

As already pointed in section 4.2.1, resource providers need to express multiple kinds of constraints on resource usage. Services/resources need to be described using a flexible framework permitting evolution. Grid4All resources will be located via a personalized, semantic rich discovery process relying on storage of metadata originating from providers and consumers (sellers and buyers of resources/services respectively) where metadata include aspects such as QoS data, resource pricing, economic models for trading resources, time/geographical location/machine allocation constraints, etc.

In general, modelling of the Grid resources (computational resources entities, storage resources entities) in the context of Grid4All project must allow their semantic description, for publishing, storing, and discovery, and must include many aspects ranging from resource configuration to performance metrics, location, availability, economic parameters such as prices, user's history, constraints (number of machines allocated, number of (sub)resources comprised by, number of providers offering specific types of resources), etc. A shared ontology providing semantic descriptions of the above aspects must be provided, enabling retrieval of resource instances based on exact or partial information about them.

Queries through a Grid4All service/resource information system should be able to search for properties of a resource in terms of concrete instance data such as ownership, location, pricing, availability, etc. For example, a) *what resources are provided by a specific provider, b) which provider has available specific resources (i.e. with a specific configuration) for a specific price and period of time?* Queries should be able to be formulated over resources' properties described using a consistent formal schema. Queries (i.e. requests for specific resources) may also involve searching properties provided by other parties such as "*find a resource that meet R functional requirements and advertised at less than a P price*". Moreover typical Grid4All users (resource consumers) should be able to receive a set of matched (exactly or partially through reasoning) offers, such that they themselves can experiment with and select one (after biding in an e-Market's auction system) that fit their needs. The query problem described in the context of Grid4All is actually a semantic matching (matchmaking) problem between services/resources' requests and offers within Grid markets.

The description and discovery of distributed and heterogeneous resources among many participants requires a common formal language for adding semantics. An ontology-based semantic description and discovery of resources will add a well defined meaning to the stored information, enabling computers and people to work in cooperation. Traditional matchmaking of resources is based on exact syntactic matching via symmetric, attribute-based matching [37]. Moreover there is a need for providing reasoning facilities for discovering also non-exact matches of queries such as subsumed matches [37]. Semantic matching using terms defined in the resource ontology allows for a loose coupling between resource and request descriptions, which removes the tight coordination requirements between resource providers and consumers [15]. For the above solutions to be integrated in Grid systems, several technologies for ontology development, storing, querying and reasoning are available. Semantic Web technology provides an adequate set of tools for choosing the most appropriate according to Grid4All services/resources ontology requirements.

Representing Grid4All Resources

A Grid Resources ontology should represent knowledge about the different types of resources that a Grid computing environment may need to exchange, manage and allocate for executing services. A resource (in the context of Grid computing) may be defined as any passive entity required for implementing functionality (i.e. Services) within a Grid. In the context of Grid4All project, resources are entities that are offered by resource providers for trading them in e-markets. We do not deal with every type of resource that a grid may allocate. In Grid4All resources are trading entities that represent computational and storage resources, either atomic or in sets (composition or aggregation). Resource consumers can use these trading entities by selecting them from a matching set resulted from their specific requests. Providers and consumers describe their orders for resources based on certain resource characteristics (e.g. type of resource, resources that is comprised by, etc) that explicitly describe resources and also based on personal preferences (e.g. price of resource that sell/buy, time to use it, number of units that need, etc.). The economic/market-related entities that need to be represented in the Grid4All resource ontology towards trading resources in a grid economy, will be discussed in the next section.

In Grid4All, resources should be able to be represented not only as atomic resources. Aggregation and composition of resources should be also represented in order to distinguish between heterogeneous and homogeneous sets (bundles) of resources:

- A *Composite Resource* is a resource composed by at least two heterogeneous resources (e.g. a CPU and Hard Disk).
- An *Aggregated Resource is a resource composed by at least two* homogeneous resources (e.g. a Cluster of compute nodes).

In the frame of Grid4All project, two main types of resources need to be represented: a) Computational Resource and b) Storage Resource:

- A *Computational Resource* is comprised by at least one CPU, one volatile memory and one operating system. A specific computational resource can reside on at least one machine (we can have one of its CPUs working on a machine and another on a different machine). A computational resource is a composite resource since it is composed by more than two other resource types i.e. it cannot be just a CPU or just a volatile memory or just an operating system.
- A *Storage Resource* can be a Hard Disk or any other type of permanent storage. A Storage Resource resides on a single machine. In case of representing a bundle of more than one (minimum of 2) Storage Resources, we introduce the entity *Storage Aggregated Resource.* Different Storage Resources that comprise a Storage Aggregated Resource may reside on different machines.

In addition to these resources, applications or services may also be needed to become available through markets, i.e. to be traded. For this reason, additional representations should be provided in the ontology as software resources representations.

A *Compute Node* is a composite resource that comprises exactly one computational resource and any number of storage resources. A Compute Node and aggregations of compute nodes i.e. Clusters, are the only tradable resources in the context of Grid4All Grid economy. A Compute Node is not an aggregated resource type since it is composed of heterogeneous resources.

The following requirements must be also met for Grid4All resources:

- Resources should be identified by a specific identification number and must be related to a specific service via which it is exposed to any (human or software) agent.
- Resources that are Hardware Resources should be described in respect to capacity attribute i.e. the unit of measurement for a resource's capacity e.g. "Mb" for a Volatile memory, "Gb" for a Hard Disk, "GHz" for a CPU.
- Hardware or Software resources should be described in respect to a Machine entity, in order to represent the allocation of resources to one or more machines. A *Machine* instance can host one or more Grid4All resources, given that they are all *Atomic resources* (they are not composite or

aggregated resources). Composite or Aggregated resources can be located in more than one machine depending on which machines their resources reside on.

- Resources should be described in respect to where they are placed, i.e. in one or more geographical locations, depending on the location of the machines they reside on.
- Resources should be described in respect to specific QoS requirements which will be discussed in a separate paragraph of this report.

Trading of Resources in Grid4All

A *Market* entity in the Grid4All ontology represents a mechanism which allows people to trade, normally governed by the theory of supply and demand, allocating resources through a *price* mechanism and *bid* matching, so that those willing to pay a price for *(buy)* a service/resource meet those willing to *sell* it. A Market instance can be started by either a resource *Seller (provider)*, a *Buyer (consumer)*, or from other third parties (e.g. a broker). Grid4All markets should be discovered by Buyers or Sellers, depending on their status (e.g. initiated, active, closed) and their type (e.g. *Consumer-initiated, Provider-initiated, bi-directional exchange market*).

Trading resources based on economic models require the representation of a marketplace and some players (market roles). The key players that drive a marketplace are: the *Providers (sellers)* and the *Consumers (buyers)*. In most related works, Consumers are represented in the market by *Brokers*, playing the key role of generating strategies for choosing providers based on their customers' (consumers) requirements. In markets where brokers are absent, this key role is transferred to consumers. The consumers need to have a *utility model*, i.e. a model that represents the way consumers demand resources and specify their preference parameters. Providers need to obtain *economic models*, i.e. models that represent mechanisms of assigning prices to resources with the aim to offer a market price at which the supply of a service equals the quantity demanded (reaching equilibrium price).

As in the conventional marketplace, the user community (buyers) specify the *demand*, whereas the resource owners' community (sellers) specify the *supply*. In the economic model, emphasis must be put on the user community and how they can influence the pricing of Grid resources (via their brokers). The resource providers and consumers interact in a competitive market environment for resource trading and access. The resource providers try to maximize their resource utilization by offering a competitive resource access cost in order to attract consumers. The resource consumers have an option of choosing the providers that best meet their requirements (exact or partial matchmaking of offers/request). Any of them (consumer or provider) can initiate a resource trading in a market like environment (consumer-initiated market or provider-initiated market) and participate in the negotiation (e.g. using an Auction) depending on their objectives. Providers should be able to invite (requests) bids from a number of providers and select those that meet their requirements (that are driven by requirements, as well as by *deadline* and *budget*). Providers can also invite bids in an *auction* by offering resources to consumers as long as their objectives are met. Both of them have their own utility model that have to be satisfied and maximized. The consumers perform a cost-benefit analysis depending on the deadline (by which results are required) and budget available (the amount of money the user is willing to invest for solving the problem). The resource owners may charge different prices for different users for the same service or it can vary depending on the specific user demands. Resources may have different prices based on the environmental influences.

In the context of the Grid4All the traded entities are Compute Node and Cluster resources. Consumers and providers specify their needs for the trading resources by placing market orders. As already pointed in section 1, such an order can be either an offer (by providers) or a request (by consumers). A request describes the resource needs (type, quantity, lease time/date) of the consumer and the price she is willing to pay for those resources. In Grid4All the minimum quality of resources expected by the consumer or provider must be explicitly specified in the request. For instance, a computing resource node can be characterized by some attributes such as CPU speed (in MHz), MIPS, L2 cache size (in Kilobyte (Kb)), Memory (in Megabyte (MB)), storage capacity (in Gigabyte (GB)), access time (in milliseconds (ms)), and data throughput (in bits per second (bits/s)). Similar resources (satisfying the above requirements) can differ in their quality, thus the offer/request matchmaking specification should provide expressiveness to allow quality specification and other, such as location and time specific, constraints.

In Grid4All market, multiple orders may be connected using an XOR (exclusive OR) connector. An XOR allows consumers and providers to order for multiple bundles. In an XOR order (request or offer) only one of the specified bundles is finally allocated. In Grid4All seems necessary to make use of XOR bids to allow consumers express multiple bundle needs. In Grid4All, consumers and providers will request for entire resources and sets of resources, not a percentage of them (for instance, they cannot request for the 40% of a Hard Disk resource). Bundle orders allow consumers and providers to indicate their needs for composite/sets of resources.

When consumers and providers have to allocate resources within a period of time, requests need to allow the specification of time intervals. In Grid4All, consumers will specify a time range within which the resources have to be allocated. In addition they need to allocate resources for some time slots. In most of the situations, the consumer may be indifferent when the allocation takes place. The consumer should only acquire the resource before a given deadline, i.e., boundaries of a time/date interval that a corresponding offer has presented. When consumers and providers need to allocate a resource during a continuous period of time, requests should allow duration specification. Time duration could be expressed in either time slots, time units (seconds, minutes, hours) or time ranges.

In many cases resources cannot be traded individually since the use of single atomic resources makes no sense. For example, the atomic resource CPU is not usable because another resource i.e. memory is needed in order to execute a task. In that case bundles are an adequate solution. Although in several systems, it could be adequate to use bundles of heterogeneous resources (composites) from different providers, for instance {CPU, Volatile Memory, Hard Disk} each one supplied by a different provider, this approach could not be useful in terms of QoS. Thus in Grid4All, a single bundle unit is represented by a Compute Node entity as an atomic unit of resource allocation through markets. A more complex bundle can be expressed as: {2 Compute Nodes satisfying attribute description = {CPU>1 GHz, mem>1 GB, disk>20 GB} for 10 time-slots (10 minutes each) between 10:00 and 18:00 where both compute nodes have to be available at the same starting time}.

Aggregated resources in a request specification are important for efficient resource allocation given that requests can be matched to equivalent resource configurations. One way to allow Consumers to request for any of the equivalent resource configurations consist in the formulation of XOR offers with multiple equivalent configurations. This choice requires the matchmaking service to search for multiple alternatives. Another way is to make use of time and space allocation units such as time-slots and number of CPU. A consumer should specify that she requires *M* time-slots of *N* CPU. Thus, different configurations apply to consumer needs.

The offer/request matchmaking specification also needs to provide flexibility in order to express conditions of trade. For example, a Consumer may need to specify whether the allocated resources come from the same Provider or from different Providers. Also it should be needed to specify that the allocation has to be completed by a specific deadline. In case that a bundle of resources has to be allocated, the minimum or maximum number of providers that may be aggregated to satisfy the resource bundle of one consumer should also be provided.

Matchmaking of resources (semantic matchmaking)

Grids join geographically distributed and/or heterogeneous computational and storage resources, and deliver them to distributed and/or heterogeneous user communities. These resources may belong to different organizations, have different usage policies and pose different requirements on acceptable requests. Grid applications, at the same time, may have different constraints that can only be satisfied by certain types of resources with specific characteristics. A user or a software agent must select resources appropriate to the requirements of the application. This process of selecting resources is called “resource matching” [15]. In a dynamic Grid environment, where resources may come and go, it is necessary to automate the resource matching to robustly meet application requirements.

For maintaining a loose coupling between resources/service requester and provider, dynamic discovery plays a crucial role [22]. Several algorithms and frameworks have been proposed to tackle this problem. Some of them are based on syntactic service descriptions, like description repository UDDI or the discovery

protocol WS-Discovery and feature symmetric and attribute-based matching of service descriptions. This is inflexible and difficult to extend to new characteristics or concepts. Others, like [39][9][24], suggest to adopt semantic service/resource descriptions for matchmaking. However, while providing semantic matching capabilities, these algorithms use centralized matching components without the employment of prices.

The Grid4All requirements for resource matchmaking are summarized in the following list:

- it is necessary to automate the resource matching to robustly meet application requirements
- adopt semantic service/resource descriptions for matchmaking
- use decentralized matching components
- employ market/price-related properties
- discover/select resources not only based on exact but also on partial information about them
- allow consumers to place any requests. They must not be aware of the exact attribute values of providers' offers in order to obtain a match.
- Reasoning engines running on resources' offer/request semantic specifications should produce a matching based on reasoning (subsumption axioms). For instance, an offer of 4 Compute Nodes will be matched with a request that is looking for a Cluster that comprises at least 2 and maximum 5 Compute Nodes.

To implement the semantic matchmaking of resources with the above requirements, an ontology specification language should be used to describe the Grid4All resources and an effective reasoning engine to compute the subsumption axioms. The automatic classification of requests (queries in the SIS) is a strong mechanism provided by reasoners for the matching of offered resource instances to the requested ones.

Examples of request/offer matchmaking

The offer/request matchmaking specification should be able to meet the above requirements and provide consistent result data. More specific it should be able to represent and reason with information as described in the following example:

Notation:

R1 = Computational Resource

R2 = Storage Resource

CR1R2 = a Compute Node comprised by R1 and R2 resources

ACR1R2 = a Cluster (an aggregated resource) comprised by CR1R2 Compute Node types.

```
Request = {(buyer: buyer-id),
        (ACR1R2, (has_Compute_Node:
                (CR1R2, (has_computational_resource:
                        (R1, (cpu-qos-attributes:
                                (cpu-speed range:[1gz, 2gz])
                                )
                        )
                 )
                (has_storage_resource:
                        (R2, (storage-qos-attributes:
                                (throughput: 40Mbs))
                                (storage-capacity: =<100Gb)
```

```
                                                    )
                                                )
                                            )
                                        )
                                    )
                                    (has_Compute_Node:
                                        At least 2
                                    )
                            )
                            (ACR1R2 requested between: (start-time: 12:00), (end-time: 18:00))
                            (ACR1R2 required for: (time-slots: 3), (time-slot-size=30mts)),
                            (ACR1R2 located at: {Place1 OR Place2}),
                            (ACR1R2 Quantity: 1),
                            (Price: 2 euros)
                    }
```

The above request looks for one Cluster resource that comprises at least 2 Compute Nodes with the following characteristics: they have a computational resource which comprises a CPU with a QoS attribute “CPU speed range” and values between 1 and 2 GHz, and they have a storage resource with “throughput” of 40Mbs and with a storage capacity of 100 Gb. The Cluster resource is requested for use between 10.00 and 18.00, for 4 time-slots and duration of 30 minutes each. The Cluster resource may be located at Place 1 or at Place 2. This implies a restriction on the location of Compute Nodes, saying that all the resources that a compute node comprises must be located at the same place.

The above request can be matched with an Offer such as the one presented in the following example:

```
Offer= {(seller: seller-id),
            (ACR1R2, (has_Compute_Node:
                        (CR1R2, (has_computational_resource:
                                    (R1, (cpu-qos-attributes:
                                            (cpu-speed range:[1gz, 3gz])
                                            )
                                    )
                                )
                                (has_storage_resource:
                                    (R2, (storage-capacity: 200Gb)

                                    )
                                )
                        )
                    )
                    (has_Compute_Node:
                        Exactly 4
                    )
            )
            (ACR1R2 offered between: (start-time: 10:00), (end-time: 18:00))
            (ACR1R2 offered for: (time-slots: 4), (time-slot-size=30mts)),
            (ACR1R2 offered at: {Place1 OR Place2}),
```

(ACR1R2 Quantity: 1),
(Price: 2 euros)
}

The above Offer will be matched to the Request described earlier although the attribute values are not exactly the same. For instance, the request asks for a Cluster with at least two Compute Nodes of a specific type, and the Offer provides exactly 4 instances of that type. The reasoner will infer that the condition of the request ("at least 2") can be satisfied by the condition of the offer ("exactly 4"). This is also the case for the ranges of the "CPU speed" QoS attribute of their CPU; one is subsumed by the other.

The matchmaking of resources' offers/requests should be able to anticipate inconsistencies of different QoS and/or resource capacity measurement units that may arise. For instance, the system should identify a matching between 1024 Mb and 1 Gb when matching storage capacity. Additionally, the system should be able to represent and/or resolve inconsistencies of other constraints that may be specified in a request/offer specification, such as the pricing scheme (what is charged and how, currency, price aggregation), the collocation or not of resources on a single machine, the number of providers that offer a single resource, the time space that a resource is offered/requested. To be able to resolve inconsistencies and infer matches between offers and requests specifications, additional models can be used to explicitly specify these constraints.

A time-related and a QoS-related ontology can prevent and/or resolve inconsistencies related to time and QoS specification respectively.

Quality of Service Ontology

Quality of Service (QoS) is any non-functional aspect that someone may use to judge quality of a service/resource.

The QoS Ontology requirement should be driven by the following two principal demands:

- Service/resource consumers aim to experience a good service performance. For example, low waiting time, good price/quality ratio, high reliability and availability towards a successfully use of the service/resource at any time.
- Service/resource providers need to formulate QoS-aware offers in order to gain the highest possible profit from their business. For example, high throughput guarantees and low response time through resource allocation, and load balancing in order to serve a high number of clients with assured quality.

The above demands are sufficient for driving our selection of concepts and metrics that should be in the QoS ontology. At the current time more information is needed in order to come up with an appropriate selection of a list of concepts that will lead us to the adaptation of an existing ontology and its potential extension.

However, at the current time high level requirements of the QoS ontology could be defined. Thus, the ontology:

a) should be extensible, i.e. custom creation of quality metrics
b) should be able to interpret different measurement units
c) should support quality measures for all the available type of resources that appear in the grid
d) should be aligned itself with other established ontologies such as OWL-Time
e) a measurement must carry information about when (e.g. time of measurement), where (locations) and how they were measured
f) a category of service/resource should be associated with a set of characteristics that affect its quality
g) the quality of a service/resource must be evaluated using a set of metrics
h) a metric must define a method to obtain measurements of a characteristic

i) a function should define how to compute a value from other values of the same characteristic (e.g. mean) or different characteristics (e.g. success consuming/total tries).

Technological requirements (for distributed and heterogeneous resource discovery and trading)

To be able to meet the above requirements, a careful selection of technologies should be performed. An ontology language able to describe Grid Resources in distributed and open environments should be selected. The language should be formal so as the semantics of defining classes and individuals to be clear, and methods for reasoning with them to be provided as well.

Towards this requirement, we should examine the capabilities of existing representation languages. More specific, we shall examine the capabilities of Semantic Web representation languages, since it is a Grid4All principle requirement for the realization of VOs and their implementation. More specific, we need to examine the capabilities of current Semantic Web KR language standards for representing Resources and/or Services. In the context of this examination, we shall compare existing technologies from many aspects, ranging from their expressive power to their performance in the large scale (many instances and classes). An important aspect of this comparison will be the reasoning facilities that these languages provide, especially at the level of facts (instances in ABox), since inference of new knowledge is rather important for the dynamic and extensible nature of resources in Grid environment.

4.3.4 State-of-the-Art of Resources/Services ontologies

Grid Resources ontologies

The OWL-S project Resource Ontology

A Grid Resources ontology has been proposed in the frame of OWL-S project [16]. The purpose of the ontology, which is stated at an abstract enough level, is to cover physical, temporal, computational, and other types of resources. This specific resource ontology sketches out the principal classes of properties a resource might have. It contains a version of the portions of the ontology that can currently be encoded in the OWL ontology language.

A basic distinction of this ontology concerns *resource types*, e.g. fuel, *resource tokens*, e.g. the fuel in the gas tank of a particular car, and *quantity*, or *capacity* of the resource token at any given instant, e.g. the five gallons of fuel in the car's tank right now. The model is primarily focused in the second of these notions. Resources in this sense can, depending on resource type, be consumed, replenished, locked, and released. A resource token, or simply *resource*, is what is available to an activity.

Resources are allocated to activities or processes. Another principal distinction of resources concerns their status after the activity stops using them. This distinction is represented as the resource's *AllocationType*. If a resource is consumable, its AllocationType is *ConsumableAllocation*. If not, its AllocationType is *ReusableAllocation*. Examples of consumable resources are food, charge in a battery, fuel, money, and time. Examples of reusable resources are the use of a device, the availability of an agent, the use of a region of space, and the use of bandwidth. A *persistent* resource can be locked and released. Many resources, such as food, are *perishable*. Thus, eating food is functional, food spoiling is dysfunctional, and eating is rapid relative to spoiling.

Resources, as described in this ontology, generally have a precise *quantitative measure of capacity* at any given instant of time. The quantitative measure might be *continuous*, such as the quantity of fuel. Or it could be *discrete*, such as the number of chairs occupied. Thus, a resource has a *CapacityType*, where the types are *DiscreteCapacity* and *ContinuousCapacity*. The capacity can be related to various other resource–related predicates, such as *activity*, *time interval*, and *time instant*. Capacities of resources can also have a *capacityGranularity*, that is, the units in terms of which the capacity is measured.

Furthermore, a resource can be *atomic*, or it can be an *aggregate* resource. Thus, *AtomicResource* and *AggregateResource* are subclasses of *Resource*. Some atomic resources can be shared by different

activities, while others cannot. Therefore, an entity *UnitCapacityResource* represents resources whose availability to an activity is a yes-no question and an entity *BatchCapacityResource* represents resources that can support multiple activities in a synchronized fashion. Both are subclasses of *AtomicResource*.

Aggregation is the act of gathering things (resources in our case) of same type e.g. Two Compute Nodes, two CPUs. Aggregated resources can be conjunctive or disjunctive: *ConjunctiveAggregateResource* and *DisjunctiveAggregateResource* are subclasses of *AggregateResource*. Conjunctive aggregated resources have to be allocated to a consumer as a unit, while one may allocate onle a subset of disjunctive aggregated resources.

Shareable resources should be understood in terms of batch capacity resources and aggregation.

This ontology satisfies many of the requirements specified in this report for a Grid4All Resource ontology. In terms of the required Grid resource ontology this ontology includes, important entities “Compute Node” and “Cluster”, the distinction of Computational and Storage resource. The distinction between Composite Computational/Storage resources and Aggregated Computational/Storage resources, are missing. In addition, several properties (stated in the requirements section) that Grid resources should have in a Grid4All environment were also missing. Finally, in terms of Grid economy, there is no specification of entities or properties that can provide/allow any kind of trading of Grid resources.

The UNICORE Resource Model

The main goal of the project UNICORE Plus (Uniform Interface to Computing Resources) was to develop production quality software for a seamless, secure, and intuitive access to the distributed resources of the German high performance computing centres. The initial UNICORE prototype implemented a simple resource model which is claimed to be still effective in practice. The resources available at each participating site - computing, data, and software resources - are made known to UNICORE by local administrators. The process is completely decentralized. Resource information which is valid at a specific time is available to all authenticated UNICORE users at the time of job creation or job submission. The users select the target system and specify the required resources. The UNICORE client is in a position to verify the formal correctness of jobs with respect to resources and alert users to correct errors immediately. For example, if the initial resource specification asks for one hour of connect time on a 512 processor T3E and the user redirects the job to a site allowing at most 256 processors, the user is informed that the job can not be executed as specified. The user is asked to adjust the values, i.e. reduce the number of processors and possibly double the connect time, or select a different site. Developments in the project EUROGRID extend the resource model to support resource brokering, cost models, and resource consumption based on application specific parameters.

The UNICORE resource model [13] [Erwin D., 2002] reflects the batch processing procedures at the UNICORE sites. For each task, resources, needed as parameters for the submission of the corresponding batch job, have to be specified. The parameters that can be specified are:

a. *number of nodes*: the number of nodes in a homogenous SMP cluster (e.g. The "SMP Cluster" consists of five rackmounted 500MHz Pentium III Xeon Quad SMP machines with a Gigabit Ethernet network)
b. *number of processors per node*: the number of processors in one SMP
c. *memory per node*: the size of memory of one SMP
d. *CPU time*: depends on the system
e. *Disk space*: consumable disk space
f. *Priority*: assign task priority, with possible values of *whenever, low, normal, high,* and *development*
g. *Requirements for particular software*, *libraries or applications:* Information resources are available to send local information to the user from a site. Software resources specify required programs and libraries. Applications are used to specify complete programs such as application packages.

This resource model (ontology) satisfies only a small set of the requirements for a Resource ontology outlined in this section, mostly at the atomic level of resource description. It is an application-oriented model rather than a general model for representing Grid resources. There are no generic semantic distinctions concerning aggregation or composition of resources, apart from the notion of a specific type of a compute *node*. More important, there is no any specification of Grid economy entities for trading the represented resources.

GLUE Schema Specification: an abstract model for Grid Resource

GLUE schema [1] is a description of core Grid resources at the conceptual level. It defines an information model, an abstraction of the real world, into constructs that can be represented in computer systems (e.g., objects, properties, behaviour, and relationships). The GLUE information model is not tied to any particular implementation and can be used to exchange information among different knowledge domains. It can also be mapped on data models that are specific of Grid Information Systems.

The core entities of the GLUE Schema are the *Site* concept and an abstraction for the *Service* concept. The site is an administrative concept used to aggregate a set of services and resources that are installed and managed by the same set of persons. The service entity captures all the common attributes to Grid Services and should be used as a base entity for the creation of service-specific schemas. The schema also provides distinction about a *Computing Element* and a *Storage Element* because of their relationship with the Site concept. In the major revision of the GLUE Schema these classes are expected to specialize (subclasses) a higher level concept such as a *Service* or a *Service group*.

The *Computing Element*, as a core concept, aims to describe the computing service that is offered to a group of users or to single users or to VOs. The Computing Element refers to the characteristics, resource set and policies of a single queue of the underlying management system. At the Grid level, computing capabilities appear as Computing Elements that are reachable from a specific network endpoint. A ComputingElement may show free job slots when jobs for a particular VO will not run due to a VO-specific quota. Conversely, the EstimatedResponseTime may be non-zero even though jobs for some VOs would start immediately. In order to deal with this possibility, Glue models the different states for different groups of users (typically on a per-VO basis or at a finer grain). This is accomplished by the VOView entity reporting state information specific to a group or a VO.

A *Cluster* is an aggregation entity for representing a complex computing resource in terms of the *SubCluster* and Computing Element entities. A cluster is a heterogeneous set of resources (computers belonging to the same cluster may have different CPUs, RAM and even different OSs), while a SubCluster is an homogeneous one. The SubCluster entity provides details of the machines that offer execution environments to jobs. It refers to a homogeneous set of hosts regarding the selected attributes. The set of attributes that are used for the summary description are present in the Host entity (an elementary computing system). A Software entity has been added in order to provide a mechanism to describe what software packages are available in the worker nodes part of the SubCluster.

Storage resources contributed to a Grid system can vary from simple disk servers to complex massive storage systems. These resources can be managed by different services, each of them taking care of a certain management aspect (e.g., data access, quota management or space management). The Storage Element is the core concept of this model and identifies the group of services responsible for the storage resource. At the virtual level, the storage resource is abstracted using the concept of Storage Area that can be made accessible to groups of users or VOs. The entities AccessProtocol and ControlProtocol are used to publish endpoints of protocols related to the storage resource. The storage area is a logical portion of storage extent assigned to VO. Storage areas can overlap the same physical space, thus having contention over the free space among different VO's. The AccessProtocol describes allowed ways to transfer files to and from a StorageElement. The ControlProtocol is similar to the AccessProtocol.

GLUE, although it provides a schema closer to the Grid4All requirement specifications for describing Grid resources, it is rather a VO-oriented model, contributing most at the level of descriptions concerning computational (Grid level computing capabilities) and communicational (publishing of protocols' endpoints related to the storage resources) entities of VOs, within and between them. More important, it does not consider any kind of economic parameters for trading the specified resources in a Grid economy environment.

A Core Grid Ontology

In the CoreGrid project, a core Grid ontology has been introduced [50] to describe fundamental Grid domain concepts and relationships. This is based on a general model of Grid infrastructures and is described in

OWL. The aim is to play a key role in the development of Grid-related knowledge bases. Its main objective is to provide a common basis for representing Grid information about Grid Resources, Grid Middleware, Services, Applications, and Grid Users.

In Core Grid ontology, the Grid is viewed as a constellation of VOs which includes VOs, users, applications, middleware, services, computing and storage resources, networks, policies of use. Resources are combined and organized by Grid middleware to provide Grid users with computing power, storage capability, and services. VOs are a container for users and applications. A VO is seen as a fundamental element of Grids. The intention is to develop a general, abstract view of Grids, towards a more general, extensible, open and VO-oriented model.

A fundamental distinction in this model is the distinction of Grid resources as *logical* or *physical*. A physical resource is a “real” resource that is part of a Grid, and the logical resource is “virtual” resource that a VO controls according to its policies, rules, and availability. Grid Middleware and Grid services are responsible for mapping logical to physical resources. From a VO perspective, a logical resource is more “realistic” than a physical, since Grids are VO-oriented.

The classes of this ontology can be grouped into three main categories:

a. VO-related: they reflect the top level of the proposed Grid model, including classes such as *VO*, *GridUser*, *GridApplication*, and *Policy*.
b. Grid Resource: Any resources including *ComputingResources*, *StorageResources*, *NetworkResources*, and *Dataset*.
c. Grid Middleware and Service: including *GridMiddleware*, functions, components, services.

Based on the above Grid model, the following types of the core classes are included in the ontology: *VO, GridResource, GridMiddleware, GridComponent, GridUser, GridApplication,* and *Service*. A GridResource can be any Hardware, Software, ComputingResource, StorageResource, NetworkResource.

This ontology is much closer to the Grid4All requirements for a Resource ontology. Although we have been re-using some of its entities (a view of the ontology entities is provided below), again, it could not be used for the key aspect of the Grid4All project, i.e the trading of resources through their offers’/requests’ matchmaking.

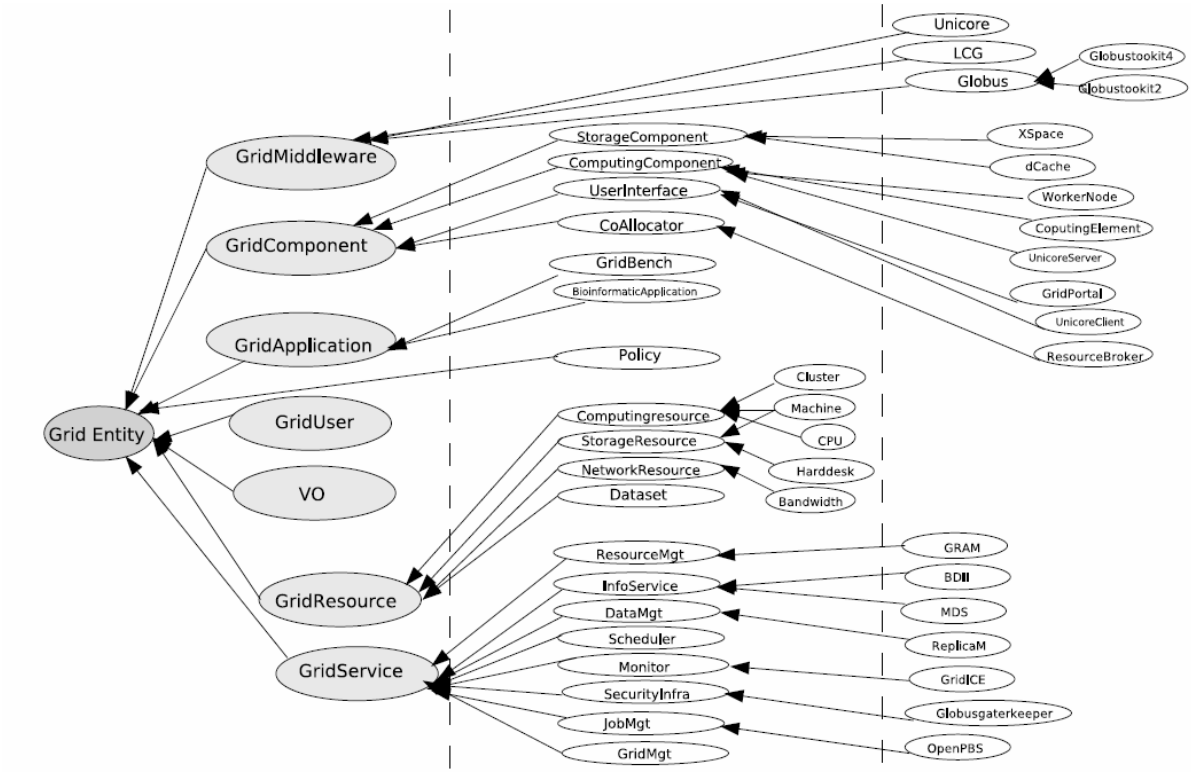


Figure 27 Part of the Core Grid Ontology

The Grid Resource Ontology

UniGridS is a follow-up project from GRIP (Grid Interoperability Project) which aims to realise the interoperability of Globus and UNICORE projects and to work towards standards for interoperability in the Global Grid Forum. UniGridS is developing a Grid Resource Ontology based on the ontology developed in the GRIP project. In the frame of UniGridS a proposal was put forward for a Grid Resource Ontology Working Group to be set up, and a number of EU projects have joined the UniGridS project to work towards a standard Grid Resource Ontology. UniGrid in collaboration with OntoGrid has developed a reference Grid Resource Ontology [33]. The ontology is coded in the W3C ontology language OWL. The GRO is intended to be developed in further discussions by the Grid community.

The basic distinctions for *Resource* and *Service* concepts have been made in GRO. *Disk, File, Network, ResourceCollection, Software, Ticket*, and *Processing* are types of Resources. *ExecutionService, LoggingService, InformationService*, and *FactoryService* are types of Services. Storage is separately defined to have types of *ClientStorage* and *ResourceStorage*. Other concepts such as *QoS, Policy, Incarnation, Agreement, Broker, Authorisation, Access, Action, Authentication,* and *Protocol* are integrated towards a Grid/VO-oriented ontology. Some of the concepts have been borrowed from Unicore and Globus projects and some other have been introduced in the framework of this project.

Although GRO ontology satisfies a good set of the Grid4All requirements for a Resource ontology, it could not be used as it is since there is no any consideration for market-related entities/properties, and matchmaking of resources' offer/request based on several economic or/and QoS, time, machine allocation (and other) constraints could not be achieved.

Market/Grid Economy ontologies

In the following paragraphs we describe the most related work, to the best of our knowledge that has been reported in the literature, which integrates economic models into Grid computing.

SHARE

In SHARE [8], the utility of one simple abstraction for global resource allocation with a number of appealing properties is explored: a centralized auction that collects user descriptions of resource configurations and the values placed on these configurations. A task of “the clearing-house” is to determine a set of winning bids and to assign appropriate subsets of global resources to individual users. The challenge with this model is the computational complexity associated with determining winners. Bidding languages are proposed that constrain the type of bids that users can make, while maintaining required expressiveness. A challenge concerning auctions presented in SHARE is the lag in clearing the auction and the uncertainty in whether resources will actually be acquired. A formulation for \Buy it Now" pricing to address some of these limitations is introduced.

A bidding language that constrains the kind of bids a user can make while maintaining required expressiveness is presented. The bidding language allows Fine-grained Resource Allocation in which bids specify the percentage of each type of resource in a bundle (40% CPU, 80% memory, 20% network, and 10% disk resources). They also permit Uniform Share resource allocation in which the bid provides partial information for a percentage of the bundle (40% share of all node resources (CPU, memory, network, disk)).

The approach cannot be applied to Grid4All because consumers do not know the exact machine configuration so that a consumer cannot decide the percentage of resources it requires.

Trading Semantic Web Services

The most recent approach related to trading of services/resources in a Grid environment is reported by Lampater and Schnizler (2006), a work funded by SEKT and CATNETS IST projects. The paper presents an architecture of an ontology-driven market for trading Semantic Web Services [22]. They report on an auction schema which is enriched by a set of components enabling semantics based matching as well as price-based allocations. The key issue of their work, in contrast to classical market mechanisms for trading Web Services, is the merging of classical auction algorithms with the semantic matching capabilities. They propose a communication language which defines how orders (requests and offers) and agreements can be formalized.

Although the presented work is very promising and contributes towards a Grid economy, (as authors admit) the existing implementations are still relying on pure syntactic matching of orders and thus not all design requirements outlined in their paper are met. Further more, they do not provide any detailed information on the ontology that they have developed. Moreover, the traded goods that are represented in the ontology are at the level of Web services and not at the level of resources.

The Grid Economy

In [4][5], the concept of Grid Economy has been introduced and an attempt to demonstrate the importance of economic models in Grid resource brokering has been conducted with several experiments. Authors have been successfully experimenting with the usage of several economic models for the resource brokering task through Nimrod/G deadline and cost-based scheduling for two different optimization strategies on the WWG (World Wide Grid) test-bed that contains peer-to-peer resources located all over the world.

This approach provided economic incentive for resource owners to share their resources on the Grid and encouraged the emergence of a new service oriented computing industry. More importantly, it provided mechanisms to trade-off QoS parameters, deadline and computational cost and offered incentive for relaxing their requirements—reduced computational cost for relaxed deadline when timeframe for earliest results delivery is not too critical. The first experiments [5] that spanned several continents showed promising results and provided a good basis for further work on Grid scheduling based on economics. Followed experiments [6] with a reference system architecture driven by Grid economy, introduced computational economy as a model for tackling challenges of resource management within large-scale Grids. The use of computational economy within Nimrod-G and Gridbus brokers for compute and data intensive applications, respectively, has been presented. Scheduling in computational and data Grids environments have been also formulated and evaluated. The results demonstrated effectiveness of commodity market-based resource allocation and also met users’ QoS requirements.

This work can be considered as the basis for understanding how resources should be traded within the Grid4All market, what are the different economic models that can be used, which are the players, what constraints can be imposed in trading, and assist us on the representation of market-related entities/properties needed within the resources ontology.

Services ontologies/languages

OWL-S

OWL-S appears as the most selected and appropriate language for modelling the Grid Services in an attempt to integrate the Web service domain and the Grid domain. OWL-S [30] is an ontology of services that has been developed as part of the DARPA Agent Markup Language Program [10]. OWL-S is an OWL-based upper level ontology for describing the properties and capabilities of Web services in unambiguous and computer interpretable way. OWL-S consist of a Service Profile for advertising and discovering services, a Service Model which support composition of services, and a Service Grounding, which associates profile and process concepts with the underlying service interface. Service Profile has functional and non-functional properties. The functional properties describe the inputs, outputs, preconditions and effects of the service. The non-functional properties describe the semi-structured information intended for human consumption, e.g. service name, service description, and service parameter. Service parameter incorporates further requirements on the service capabilities, e.g. security, quality-of-service (QoS), geographical scope etc. Service Grounding enables the execution of the concrete Web service by binding the abstract concepts of the Service Profile and process to concrete messages. The main parts of the OWL-S upper ontology are shown in Figure 28.

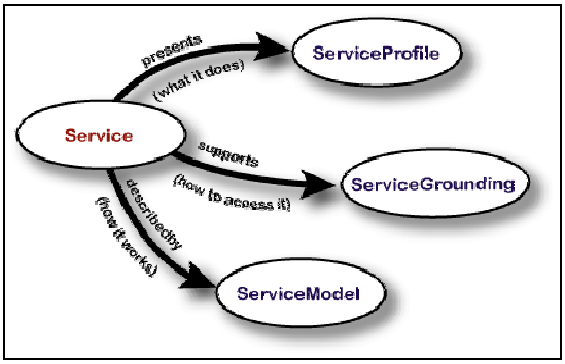


Figure 28 Main Concepts of OWL-S ontology

Extending OWL-S with Grid Jobs

In the context of the CoreGRID [60] project a work [4] has been involved with the convergence of the Grid towards Web services and the proposal of an extension of the OWL-S ontology with Grid related concepts of Grid Services, Job and Job Submission Description. The work aims to identify relationships between the basic concepts of Web Services and the Grid. In general, a Grid Service is a Web service that has stateful interactions, life-cycle management, externally accessible state and the possibility to subscribe for notification of events. A Job is an activity that is executed by a Grid service, and a Job Submission Description is a unit of work describing a Job. Typically, a user submits a Job Submission Description to the Grid for having the job executed.

Figure 29 depicts the extended OWL-S structure introducing the Grid related concepts. The concept GridService is a specialization of the class Service. The JobSubmissionDescription is submitted by a Resource and describes a Job. JobSubmissionDescription is part of process but not Process itself as the execution of a Job is. Therefore, as can be seen in Figure 30 Job is a specialization of an AtomicProcess (means that can be invoked) and can be executed by an instance of a GridService.

Converting WSRF to OWL-S

In the context of KW-fGrid [21] project a tool that semi-automatically generates the OWL-S description of both stateless and stateful services from their WSRF documents has been developed. The derived service ontology, which is responsible for describing the Grid service, in most of its part is based on the OWL-S standard. Some domain specific concepts are added and extend OWL-S only when needed.

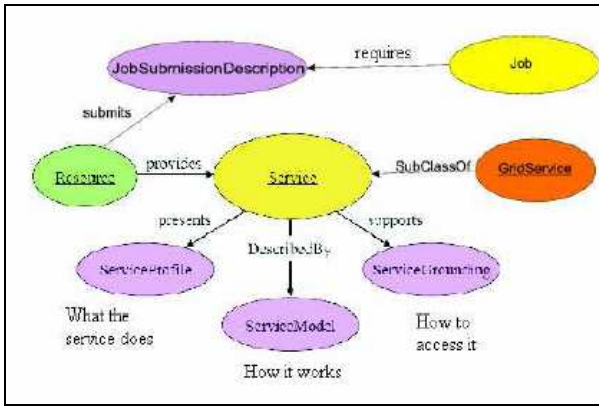


Figure 29 The proposed concepts 'Job Submission Description', 'Job' and 'Grid Service' in the OWL-S ontology.

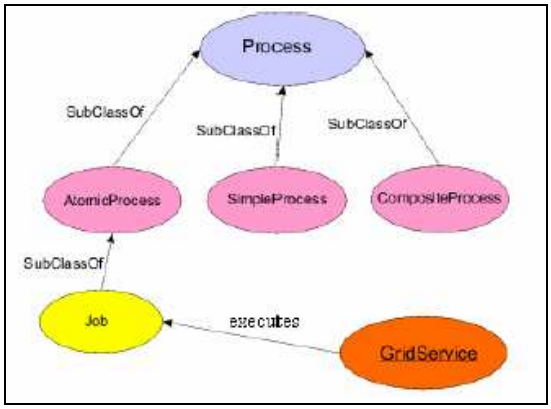


Figure 30 The 'Job' concept is specialization of the Atomic Process Concept.

This works comes up with some interesting conclusions and assumptions: No major differences between stateless and stateful services exist in the modelling of their semantics using the OWL-S. Also, it has been noticed that the resource properties, as defined in WS-Resource specification can be dynamic, i.e. deletion and addition of properties on the fly, thus the representation of such a complex conceptualization was avoided and assumed that they work only with resources that have static properties. Finally, it was correctly observed that a Grid service may have multiple instances of the same service hosted by different servers. Therefore, multiple OWL-S groundings should be introduced.

OWL for Workflow and Services (OWL-WS)

In the context of NextGrid [3] project a work towards modelling Abstract and Concrete Services and Workflows took place. OWL-S selected as the most appropriate language firstly because of its ability to express control and data flows and secondly because it has been developed on top of standard semantic web technology. The complete OWL-S model is adopted as it is in OWL-WS to represent Concrete Services and it is easily adapted to represent Concrete Workflows (see Figure 31). More details concerning the model of OWL-WS can be found in [3]. A Concrete Service is modelled as Service with its own Profile, Process and Grounding whose elements refer to a single implementation. On the other hand, Concrete Workflow is modelled as a Service with its own Profile, Process and Grounding whose elements refer to multiple implementations.

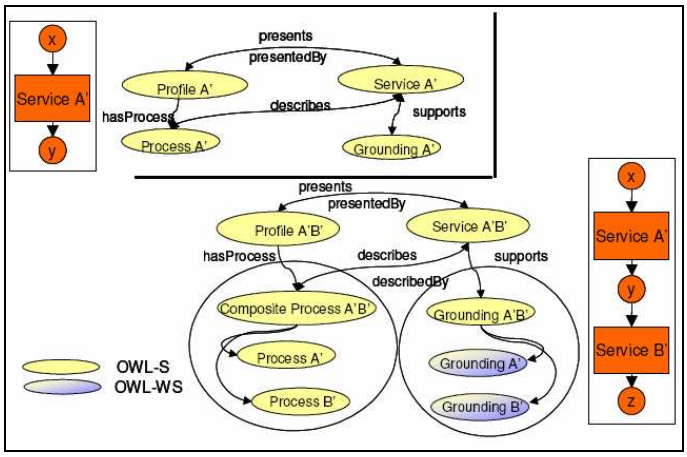


Figure 31 OWL-WS: concrete components representation example.

OWL-WS use the concept 'Composite Process' for modelling workflows. Moreover, OWL-WS defines the concept of 'Abstract Process' which is an Atomic Process having no link to any Grounding and provided with a new property named 'definedBy' which points to a Profile. Using this extension the Abstract Service and Abstract Workflow can be defined. An Abstract Service is a Service with its own Profile and Abstract Process with the 'definedBy' property pointing to the Service Profile itself. An Abstract Workflow is a Service with its own Profile and Composite Process. Each Composite Process is an Abstract Process with is own Profile referred by the 'definedBy' property. Figure 32 depicts the above.

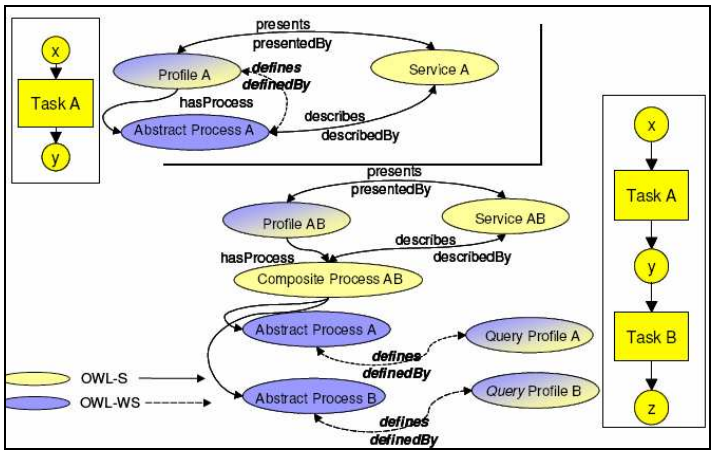


Figure 32 OWL-WS: abstract components representation example.

Moreover, OWL-WS supports Service Grouping management by providing information not only for the single Abstract Service but also for specific service groups modelled as internal workflows. Figure 33 depicts this capability.

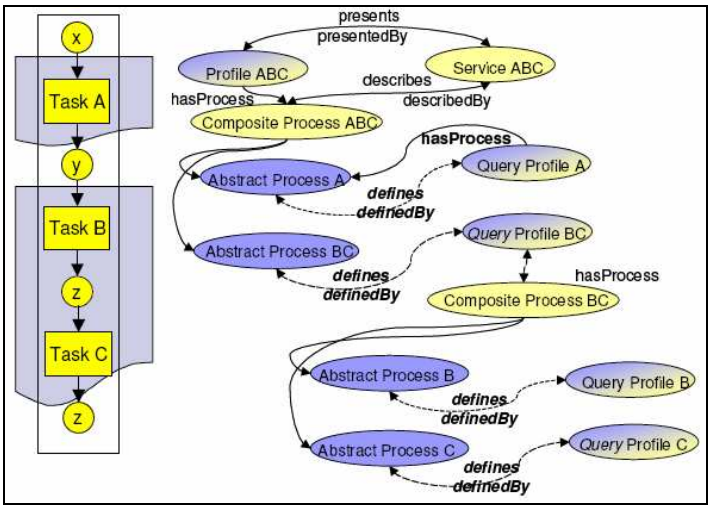


Figure 33 OWL-WS: service grouping capability example

WSRF-S

Web Service Modelling Language is a language for the specification of ontologies and different aspects of Web services. It provides a syntax and semantics for the Web Service Modelling Ontology [47] and uses well-known logical formalisms in order to enable the description of various aspects related to Semantic Web Services. WSMO working group investigated whether and how WSRF could be combined with semantic technologies [49] in the context of WSMO and concluded that any semantic extensions to WSRF will be "only" applications of semantic extensions to XML Schema and XML querying technologies [49].

4.3.5 QoS ontologies

Quality of Service Ontology

Quality of Service (QoS) is any non-functional aspect of the system that someone may use to judge quality of a service that exists in that system.

The QoS Ontology requirement should be driven by the following two principal demands:

I. Service consumers aim to experience a good service performance. For example, low waiting time, good price/quality ratio, high reliability and availability towards a successfully use of the service at any time.
II. Service providers need to formulate QoS-aware offers in order to gain the highest possible profit from their business. For example, high throughput guarantees and low response time through dynamic resource allocation, and load balancing in order to serve a high number of clients with assured quality.

The above demands are sufficient for driving our selection of concepts and metrics that should be in the QoS ontology. At the current time more information is needed in order to come up with an appropriate selection of a list of concepts that will lead us to the adaptation of an existing ontology and its potential extension.

However, at the current time high level requirements of the QoS ontology could be defined. Thus, the ontology a) should be implemented in OWL-DL, b) should be extensible, i.e. custom creation of quality metrics, c) should be able to interpret different measurement units, d) should support quality measures for all the available type of resources that appear in the grid, e) should be aligned itself with other established ontologies such as OWL-Time. Also, f) a measurement must carry information about when (e.g. time of measurement), where (locations) and how they were measured g) a category of service should be associated with a set of characteristics that affect its quality h) the quality of a service must be evaluated using a set of metrics, i) a metric must define a method to obtain measurements of a characteristic j) a

function should define how to translate a value from other values of the same characteristic (e.g. mean) or different characteristics (e.g. success consuming/total tries).

DAML-QoS

In [52] a DAML-QoS ontology is provided as a complement for DAML-S service ontology to provide a better QoS metrics model since DAML-S (and subsequently OWL-S) does not provide a detailed set of classes, properties and constraints to adequately describe QoS.

DAML-QoS consists of three layers: (a) The QoS profile layer, which is designed for matchmaking purposes, (b) the QoS property definition layer for elaborating the property's domain and range constraints and (c) the metrics layer for metrics definition and measurement. Additional specific properties can be added to the ontology for various service categories such as storage and computational services. The DAML-QoS ontology's core concepts along with specific defined properties are described in [52]. It is worth noticing that DAML-QoS permits a Service Profile to have multiple QoS profiles. The ontology also contains a basic profile for describing a service according to a system-centric QoS. This basic profile contains response time, cost, reliability and throughput metrics.

- Response Time is defined as the total time needed by the service requester to invoke the service. It is measured from the time the requester invokes the service to the time the requester finishes with the service usage.
- Cost represents the cost associated with the execution of the service. It is necessary to estimate the guarantee that financial plans are followed. The cost can be further divided into more refined components such as service execution cost and network transportation cost.
- Reliability corresponds to the likelihood that the service will perform when the user demands it and it is a function of the failure rate. Each service has two distinct terminating states: one indicates that a web service has failed or aborted; the other indicates that it is successful.
- Throughput represents the number of Web service requests served at a given time period. It is the rate at which a service can process requests.

QoSOnt

QoSOnt [12], is an ontology for QoS implemented in OWL formalism and aims mainly at Web services. The ontology consists of a set of interconnected smaller ontologies. Figure 34 depicts the different layers of the ontology.

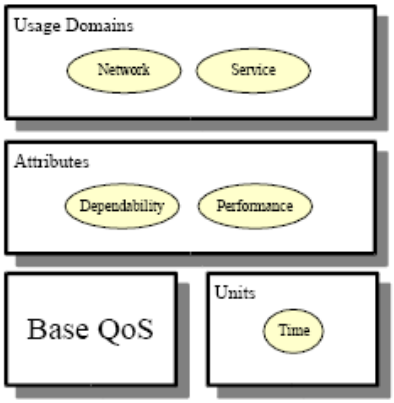


Figure 34 Layers of the ontology

QoSOnt represents many of the commonalities discovered between other QoS specification languages examined. Unlike most of these languages however, QoSOnt also aims to be generic enough to be used no matter what one's particular view of QoS is. This approach has been to provide a base set of useful constructs which cover common cases. The architecture is designed to allow third parties to replace parts of the ontology as needed. For instance, one may have a different view of dependability, or have produced a

time ontology which suits specific purposes better. Obviously this is only useful if the relevant ontologies are shared with the community they wish to interact with.

The base QoS layer contains generic concepts relevant to QoS. Unit ontologies also logically reside in this layer. Time is the most relevant unit in QoS, and is the only unit ontology defined at the moment. It represents units of time and how to convert between them. This means that an inference engine could establish, for instance, that 1 minute is the same as 60,000 microseconds. This is particularly useful if clients use the same metric as providers - but different units.

The attribute layer contains ontologies defining particular QoS attributes and their metrics. On top of this is the domain-specific layer, which links the lower layers to specific types of a computer system. For instance, the network ontology defines that certain QoS attributes are specific to a particular network route and the service ontology that QoS attributes sometimes refer to particular services, service operations, etc.

OWL-QoS

In [27], a QoS ontology lets service agents match advertised quality levels for its consumers with specific QoS preferences. Providers express policies and consumers express preferences using the QoS ontology. This also enables the consumers to configure service proxy agents so that they have the necessary behaviours to monitor and record consumer and service interactions. It helps to distinguish three ontologies for QoS: upper, middle, and lower. The upper ontology captures the most generic quality concepts and defines the basic concepts associated with a quality, such as quality measurement and relationships:

- Quality represents a measurable non-functional aspect of a service within a given domain. Quality attributes relate to each other.
- QAttribute captures a given quality's type, for example, whether it's a monotonic float attribute (a quality whose value is a floating point number and increases if the value reflects improvements in the quality).
- QMeasurement measures a Quality objectively or subjectively. Agents make objective measurements automatically, whereas subjective measurements involve humans. A measurement has a validity period and can be certified.
- QRelationship describes how qualities are correlated. Service response time, for example, could be negatively correlated to throughput. Such quality relationships often reflect the trade-offs providers make in their service implementations. Qualities are potentially related in terms of direction (opposite, parallel, independent, or unknown) and strength (such as weak, mild, strong, or none).
- AggregateQuality is a quality composed from other qualities. The price/performance ratio, for instance, aggregates price and performance.

The QoS middle ontology incorporates several quality aspects encountered in distributed systems:

- Availability is the probability that a service responds to consumer requests. It has two subclasses: MTTR (mean time to repair, meaning the average time for restoring a failed service) and UpTime (the duration for which the service has been operational continuously without failure). Availability is mildly parallel to reliability and typically mildly opposite to capacity.
- Capacity is the limit on the number of requests a service can handle. When a service is operated beyond its capacity, its availability and reliability are negatively affected.
- Economic captures the economic conditions of using the service. Usage cost is a key economic attribute.
- Interoperability is the ease with which a consumer application or agent interoperates with a service. It defines, for example, whether the service is compliant with a specified standard, such as the WS-I Basic Profile, or specific versions of standards like WSDL.
- Performance characterizes performance from the consumer's perspective. Examples are Throughput (the rate of successful service request completion) and ResponseTime (the delay from the request to getting a response from the service).
- Reliability is the likelihood of successfully using a service. Typically, it parallels availability, but its main aspects also include Fault Rate (the rate of invocation failure for the service's methods); MTBF (mean time between failures); Consistency (the failure rate's lack of variability); Recoverability (how well the service recovers from failures); Failover (whether the service employs failover resources, and how quickly); and Disaster resilience (how well the service resists natural and human-made disasters).
- Robustness is resilience to ill-formed input and incorrect invocation sequences.
- Scalability defines whether the service capacity can increase as needed.

- Security captures the level and kind of security a service provides. Its key components include Auditability (the service maintains auditable logs); Authentication (the service either requires user authentication or accepts anonymous users); Encryption (the type and strength of encryption technology used for storage and messaging); and NonRepudiation (whether consumers can deny having used the service).
- Integrity is a measure of the service's ability to prevent unauthorized access and preserve its data's integrity.
- Stability is the rate of change of the service's attributes, such as its service interface and method signatures.

QoS Ontology Language

A recent attempt [32] aims to provide a standard model to formally describe arbitrary QoS parameters using the OWL formalism. The ontology designed towards satisfying rich set of requirements, such as expressiveness, robustness, flexibility, accuracy, scalability, performance, completeness, user and developer-friendliness, and interoperability.

The derived ontology acts as an upper level ontology for specifying concrete QoS ontologies. Therefore, appropriate development is needed towards defining the concepts that will be involved in the measuring of a service's quality. An interesting feature of this upper ontology is the definition of conversional formula that is responsible for converting the measurement units of a metric.

4.3.6 Policy ontologies

In the context of this report, a *policy* is a set of non-functional constraints and capabilities that a Web service has and allows itself to specify requirements for initiating the interaction between the service requester and itself. For example, the functional capability of a hotel booking service is to reserve hotels' rooms, whereas a non-functional capability of the service may be that it gives final reservations within 24 hours of the request and that it supports strong encryption for communication. Moreover, a non-functional requirement of a service that uses a specific printer to print documents may be that only persons that belong to a specific group (i.e. senior developers) can have print access to it.

A policy ontology-language should address an *authorization policy*, which is whether a requester is permitted to or denied to access to a particular resource/service and *obligation policies*, which is what and when a requester is required to do to a resource/service. Also, it should address *usage policy*. That is what constraints should satisfy the mean that utilizes the resource/service, for instance the service A should run on a linux-styled operating system with memory not less than 500 MB. Moreover usage policy may also dictate some obligations that a resource/service consumer should follow, for instance the service consumer must not use the output of the service in a commercial use. And last but not least, the policy ontology should be OWL-DL compliant, extensible to custom polices and capable to combine already defined policies using the operators and, or and not.

All the following reviewed policy languages except WS-Policy, satisfy more or less the above requirements. However more specific requirements concerning the exploitation of policy ontology in the context of grid4all project should be stated in order to choose the most appropriate one.

It should be also stated that in a grid environment policies should be used in other levels. For example at provider level, e.g. how many resources can be allocated to different purposes, and the Virtual Organization (VO) member's level, e.g. the resource providers or a VO member may not participate in a VO unconditionally, a member of a particular VO should satisfy some criteria or a service should not scheduled in VO if a QoS measure is below a threshold.

WS-Policy

WS-Policy (WS-Police, 2006) is a policy language being developed by IBM, Microsoft, BEA, and other web services vendors in order to provide a general purpose model and syntax to describe and communicate the

policies of a Web service. Policies consist of assertions and alternatives and are expressed in XML constraint on a grammar in XML Schema. An assertion is the basic unit of a policy and defined by a Qualified Name which point to a domain-dependent entity. Also, an assertion can contain nested policy expressions. An alternative is a set of assertions and a set of alternatives comprises a policy. All assertions in an alternative have to be satisfied by the requester of a service that follows that policy - alternative.

Although, this model has the most momentum of the existing one, lacks semantic expressivity and reasoning capability. Therefore, is by the beginning unable to support our requirements.

Rei Policy Specification Language

Rei [18][19] is a policy specification language which is modelled on deontic concepts of permissions, prohibitions, obligations and dispensations and is able to describing a large variety of policies ranging from security policies to conversation and behaviour policies. Polices are defined by declaring what an entity can/cannot do and what it should/should not do in terms of actions, services, and conversations.

Rei policy language consists of domain independent ontologies which includes the concepts, e.g. permissions, obligations, actions, speech acts, that are used for declaring a policy. Policies are specified as constraints over allowable and obligated actions on resources of the environment. Domain specific ontologies are used to conceptualize the resources and their properties of the environment. Rei also includes logic-like variables giving it the flexibility to specify relations like role value maps that are not directly possible in OWL.

A very interesting feature of Rei policy language is the meta-policy specifications for conflict resolution. Conflicts may occur between permissions and prohibitions, obligations and prohibitions, and obligations and dispensations. Rei defines two meta-policies for resolving conflicts: a) setting the modality precedence and b) stating the priority between rules within a policy or between policies. In the former meta-policy it becomes well known what policy overrides what policy and in the latter a priority setting ranks the rules within polices and the policies between them.

KaOS Policy Ontology and Services

KaOS Policy Ontology and Domain Services [41][42] use an ontology, which is formalized in OWL, to build policies. KaOS's ontology is provided containing domain independent concepts necessary to describe a policy. The main concepts of the ontology are: actors, actions and context. A context can refer to objects, such as domain specific entities, computing resources etc., which are the target of the action. The provided ontology can be further extended with domain specific concepts.

The KaOS ontology distinguishes between authorizations and obligations. Authorization is divided into positive and negative *authorizations*, *which are* constraints that permit or forbid some action respectively. Obligations are divided into positive and negative, which are constraints that require some action when a state- or event-based trigger occurs or that serve to ignore such a requirement. A policy is an instance of one of following four basic policy concepts: *PositiveAuthorization*, *NegativeAuthorization*, *PositiveObligation*, or *NegativeObligation*. The property values determine management information for a particular policy, e.g its priority. The type of policy instance determines the kind of constraint KAoS should apply to the action, while a policy's action class is used to determine a policy's applicability in a given situation. The *Action* concept exploits OWL restrictions to narrow *scopes-of-action* properties and is associated with actors. KaOS use Java Theorem Prover (JTP, 2004) in order to identify and harmonize conflicting policies.

Mapping WS-Policy to OWL-DL

The lack of formal semantics in one of the leading policy languages, named WS-Policy, drove Kolovski et al., [20] to map WS-Policy descriptions into OWL-DL formalism. WS-Policy has shown to be an expressive subset of OWL-DL.

OWL-DL proved an interesting framework for exploring richer policy languages with minimal implementation cost. Also, expressing policies in OWL-DL enabled harmlessly the exploitation of inference capabilities.

Thus, policy inclusion (if x meets policy A and A is sub-class of B then x also meets B), policy equivalence (police A is equivalent to policy B), policy incompatibility (if x meets policy A and not policy B then A and B are disjoint), police incoherence (nothing can meet policy A, it is unclassifiable) and policy conformance (x meets policy A, x is type of A) got in sight of transformation to OWL-DL.

4.3.7 Resources/Services matchmaking

Matchmaking has been a hot topic of MAS (Multi-Agent Systems) research, related to question on how to find a suitable agent for a specific problem. The matchmaking in MAS involves semantic service matchmaking using the concept relationships and word distance to determine the semantic similarities of advertisements and requests. The matchmaking in MAS does not involve other resource types and the matchmaking results are exact, i.e., only "true" and "false" are allowed.
Research for service/resource discovery in the Internet involves ontology-based matchmaking. The traditional methods of service discovery include name matchmaking and keyword matchmaking. Some new methods are based on ontologies. In [34] a semantic matchmaking framework based on DAML-S, a DAML (DARPA Agent Markup Language)-based language for service description, was proposed for semantic matchmaking of web services capabilities. The basic idea is that an advertisement matches a request when the service provided by the advertisement can be of some use to the requester. The matchmaking is performed on the outputs and inputs of the advertisement and the request is based on the ontologies available to the matchmaker. Through the subsumption relationship of one concept of the input/output of the advertisement and one concept of the input/output of the request, three levels of matching can be determined: exact, subsume, and fail. The idea of checking the concepts of input and output is similar to the one in the MAS research.

In the following paragraphs we describe some of the related works concerning resource/service matching at the semantic level towards an effective resource/service discovery in Grid environments. We conjecture that these works are closely related to the Grid4All requirements for an ontology-based resource discovery service.

BondGrid

In an intelligent grid environment, the BondGrid [2], a resource matching scheme is proposed that supports a variety of matching functions including fuzzy ones. Given a request and a set of resources, the goal is to find a set of resources that best match the request. A *request* is a (*n*+1)-tuple consisting of *n* attributes (a_1, a_2, …, a_n) and a function of these attributes to be evaluated in the context of resources, i.e., request = [a_1, a_2, ..., a_n, $f(a_1, a_2, ..., a_n)$]. An attribute of a request is a mapping from an attribute name to an attribute expression. A *resource* is an *m*-tuple consisting of *m* attributes (a_1, a_2, ..., a_m). The resource that returns the largest value of function *f* is considered as the one that best matches the request. Attribute names are constructed according to the corresponding resource ontologies. Resource ontologies are a critical component of the matchmaking framework.

For a request = [a_1, a_2, ..., a_n, $f(a_1, a_2, ..., a_n)$], the function *f* is an expression that is the combination of attribute expressions $f_1(a_1)$, $f_2(a_2)$, ..., and $f_n(a_n)$ through mathematical and/or logical operators, where $f_1(a_1)$, $f_2(a_2)$, ..., and $f_n(a_n)$ are to be evaluated in the context of the corresponding attributes of the resource. The expression for function *f* may involve:

1. Boolean expressions can be combined with the use of Boolean operators "&" and/or "|".
2. Arithmetic expressions can be combined with the use of arithmetic operators, such as "+", "-", "*", and "/".
3. Fuzzy expressions can be combined with the use of fuzzy operators "&&". The evaluation result of multiple fuzzy numbers connected by "&&" are the average of these fuzzy numbers.
4. A Boolean expression can be combined with an arithmetic expression or a fuzzy expression through the Boolean operator "&". If the Boolean expression returns 1, they are evaluated to the value returned by the arithmetic expression or the fuzzy expression. If the Boolean expression returns 0, they are evaluated as 0.
5. Expressions are combined through "*if*", "*then*", and "*else*" constructs.

Authors define three types of matching functions. A matching function *f* may contain Boolean expressions and return a Boolean constant (“true”, 1 or “false”, 0). *f* may also contain arithmetic expressions and return a positive real number. They also allow *f* to contain fuzzy expressions and return a fuzzy number in [0, 1]. The higher the returned value, the better a request can be satisfied.

The matchmaking framework includes a resource specification component, a request specification component, and matchmaking algorithms. A request specification includes a matchmaking function and possibly two additional constraints, a *cardinality threshold* and a *matching degree threshold*. The cardinality threshold specifies how many resources are expected to be returned by the matchmaking service. The matching degree threshold specifies the least matching degree of one of resources returned by the service.

Although the approach builds on ontology-based resource matching, the need for following specific rules for constructing attribute names according to the corresponding resource ontology is rather restrictive and could not be applied in open and distributed Grid environments outside organization boundaries.

Ontology-based resource matchmaking (OMMS)

In OMMS [15] ontology-based matchmaker, authors employ a flexible and extensible approach for performing Grid resource selection that, unlike the traditional Grid resource selectors, uses separate ontologies to declaratively describe resources and job requests. Instead of exact syntax matching, the ontology-based matchmaker performs semantic matching using terms defined in ontologies. The loose coupling between resource and request descriptions remove the tight coordination requirement between resource providers and consumers. In addition, the matchmaker can be extended by adding vocabularies and inference rules to include new concepts about resources and applications, and adapting the resource selection to changing policies.

In previous work of OMMS, authors have designed and prototyped the matchmaker using TRIPLE to use ontologies encoded in W3C’s Resource Description Format (RDF) and rules (based on Horn logic and FLogic) for resource matching. Resource descriptions, request descriptions, and usage policies are all independently modelled and syntactically and semantically described using RDF schema. They utilize inference rules for reasoning about the characteristics of a request, available resources, and usage policies to appropriately find a resource that satisfies the request requirements.

A request is expressed using the request ontology. A persistent Grid matchmaking service has been designed that can support multiple clients simultaneously. Since there is only one single TRIPLE instance that handles all requests, authors have implemented a synchronization mechanism to ensure an atomic operation for each request. The granularity of these operations is chosen so that the waiting time for each operation is short. Clients submit requests in RDF, and receive a list of matching resources either in an outline form (e.g., resource names) as strings or in a detailed form (e.g., resource names and their capabilities) as RDF. Users can express their preference for matched resources by using a ranking function. The ranking function is an arithmetic expression expressed in terms of resource properties; for example, 10_CPUClockSpeed+PhysicalMemory. Clients can indicate the number of returned resources which will be sorted based on their ranking values.

As a key part of OMMS, the resource discovery component dynamically collects resource information from multiple sources, transforms the information into the resource ontology, and updates the backend knowledge base accordingly. Since different Grid resource providers might express their capabilities using different schemas and encoding mechanisms (e.g., XML, LDAP), an ontology translator is developed to translate the various formats used in a heterogeneous Grid environment into OMMS’ resource ontology. The design goals of this module include collecting various categories of resources and updating with least communication overhead without service interruption.

According to the authors, the existing system is designed to be application or domain independent. Higher-level application models can be built on top of the existing system. As an ongoing research, they plan to expand the request ontology to include application-level description as well as their performance models, allowing users to describe their requirements in their own domain specific terms. Depending on the request, additional information might be needed. For example, the sizes of input files are needed to infer the disk

space requirement associated with the computing resource. In addition to the ontology expansion, authors plan to extend the matchmaker service to dynamically gather additional information from other knowledge sources (such as the Metadata Catalog Service) and incorporate that into the matchmaking procedure.

Knowledge layer on Gridbus broker

This work [37] addresses the need for a semantic component in the grid environment to discover and describe the grid resources semantically. It proposes a semantic grid architecture by introducing a knowledge layer at the top of Gridbus broker architecture and thereby enabling brokers to discover resources semantically. The semantic component in the knowledge layer enables semantic description of grid resources with the help of an ontology template. The Ontology template has been created using Protégé-OWL editor for different types of computing resources in the grid environment. The Globus Toolkit's MDS is used to gather grid resource information and Protégé-OWL libraries are used to dynamically create a knowledge base of grid resources. Algernon inference engine is used for interacting with the knowledge base to discover suitable resources.

In grid environment where resources are generally owned by different people, communities or organizations with varied administration policies and capabilities, obtaining and managing these resources is not a simple task. Resource brokers simplify this process by providing an abstraction layer for users to access heterogeneous resources transparently. Gridbus broker is a resource broker designed to support scheduling of both computational and data grid applications. However, the resource discovery module implemented in the Gridbus broker does not support semantic description and discovery of grid resources, as it uses the Globus Grid Index Information Services (GIIS) or Grid Marker Directory (GMD) to gather grid resource information. In this paper, authors propose a knowledge layer on the top of Gridbus broker architecture for semantic description and discovery of resources.

Authors proposed a five layered architecture that implements a knowledge layer on top of Gridbus broker as shown in the following figure and it can be used for building semantic grid infrastructure.

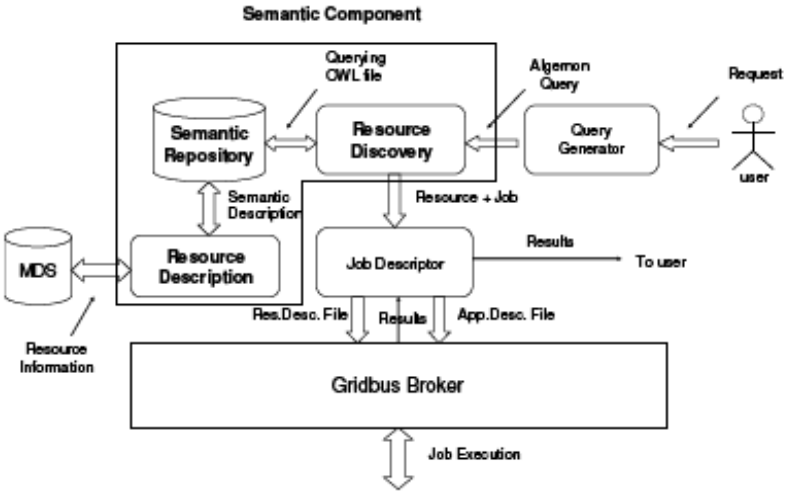


Figure 35 The knowledge layer on top of Gridbus broker

The Resource Discovery Module allows the users to submit the information about required resource to execute their job. It then generates appropriate Algernon query depending on the requirements specified by the user and executes these queries over the ontology knowledge base to obtain best possible resources closely matching to the request. The discovery module retrieves resources that exactly match with that of request. It also retrieves resources subsumed by the query when exact matching is not found. Once suitable resource is obtained, the resource discovery module submits the resource information along with the user's

job to the job descriptor. With this information, the job descriptor creates an application description file and resource description file. Both these files are required by the broker to successfully run the application in the specified resource. The broker executes the job on the specified resource and generates results. The discovery module aggregates the results from the broker and delivers to the user.

Protégé has been installed in one of the machines (which don't have to be a resource provider) and an ontology-driven template has been created by considering different computing resources in the grid. The concept of these resources has been defined properly using relations and properties so that the characteristics of any resource can be defined. The MDS component offers a tool to retrieve resource information which is accessed through remote machines. The grid-info-search tool aggregates various properties of grid nodes and stores them in the LDAP server. These are retrieved using suitable LDAP queries. The resource description module is developed using Java programming language. The module accesses the grid nodes and retrieves resource information by executing LDAP queries on those nodes and updates into the ontology template. The Protégé editor offers versatile libraries called Protégé-OWL APIs with which one can manage the ontology and perform several operations over the ontology such as creating and deleting the instances of concepts, assigning values to the properties etc. For every type of information retrieved from the grid node, instances of appropriate concept in the ontology template are created, forming conformity relation between instances and their respective concept types. Further, an instance will be exactly related to only one concept type. Also, the values of various properties retrieved are assigned to respective properties of the appropriate concepts in the ontology template. This semantic description of resources facilitates the use of an inference engine to interact with the knowledge base and retrieve information semantically. Moreover, the description module is made to execute periodically so that addition and removal of resources is accounted in the knowledge base dynamically.

Although an important recent implementation of Grid resources semantic matchmaking has been presented within this related work, there is an important restriction: the ontology template developed depends on MDS component and hence it may not support middleware other than Globus.

Semantic approach to service discovery in a Grid environment

In [26], a matchmaking framework for service discovery in Grid environments is proposed, based on three selection stages which are context, semantic and registry selection. It provides a service discovery process by using semantic descriptions stored in ontologies which specify both the Grid services and the application knowledge. The framework permits Grid applications to specify the criteria a service request is matched with and enables interoperability for the matchmaking process. A prototype implementation is presented, and an enhancement of the matchmaking process is achieved with a similarity metric which allows quantifying the quality of a match.

Related projects are trying to overcome the interoperability problem which Grid systems face, however, are concerned with applying semantics to resources in order to have a more powerful matchmaking technique. The approach proposed in this paper is concerned with application-level issues and requirements (also myGrid project[5]). The main requirements which have driven the development were high degree of flexibility and expressiveness, support for subsumption and data-types and a flexible and modular structure implemented with the latest Web technologies. The main difference to the approaches proposed by others is the concept of a three step discovery process consisting of application context selection, services selection and registry selection. It allows the capture of the application and Grid services semantics separately and it supports application developers and Grid services developers to register application and services semantics separately. For the discovery process, this separation allows a classification of the application semantics in order to find service descriptions in the Grid services ontology.

The matchmaking process is designed with respect to the criteria listed below:

1. *High Degree of Flexibility and Expressiveness*

   Different advertisers would want to describe their Grid services with different degrees of complexity and completeness. The description tool or language must be adaptable to these needs. An advertisement

[5] http://www.mygrid.org.uk/

may be very descriptive in some points, but leave others less specified. Therefore, the ability to express semi-structured data is required.

2. *Support for Subsumption*

Matching should not be restricted to simple service name comparison. A type system with subsumption relationships is required, so more complex matches can be provided based on these relationships.

3. *Support for Data Types*

Attributes such as quantities should be part of the service descriptions. The best way to express and compare this information is by means of data types.

4. *Matching Process should be Efficient*

The matching process should be efficient which means that it should not burden the requester with excessive delays that would prevent its effectiveness.

5. *Appropriate Syntax for the Grid*

The matchmaker must be compatible with Grid/Web technologies and the information must be in a format appropriate for a Grid environment.

6. *Flexible and Modular Structure*

The framework should be flexible enough to allow Grid applications to describe their context semantics and Grid services to describe their service semantics in a modular manner.

7. *Lookup of Matched Services*

The framework should provide a mechanism to allow the lookup and invocation of matched services.

Depending on the matching modules and the defined application and services ontologies, a semantic match is performed. Every pair of request and advertisement has to go through several different matching modules of the matchmaking process. The final match with the service registry is performed in the registry module. Information is provided to the service requester by sending contact details and related capability descriptions of the relevant service provider.

The context matching module allows matching of the service request by means of context semantics defined in the application ontologies. The application software of the different HEP (High Energy Physics) applications specifies the service request within their own application context. In this module a mapping from an application service request to a context-based service request is performed. It contains the concept of the application domain specified by classes, datatypes and properties. The matching engine comprises a DAML parser, an inference engine and a defined set of rules in order to reason about the ontologies.

Authors report performance problems related with the complexity of the ontology and rules and suggest that a faster reasoning process is desirable and needs to be investigated. Also, the service description contains prepositions or articles such as to, in, of, a etc., which need to be removed as they do not express the functionality of the service and only distort the similarity value. Furthermore, not every attribute or description has the same expressiveness and should be ranked with a different weight value accordingly. This implies that human intervention for the ranking process becomes necessary which is a drawback for the automation of the matchmaking process. Finally, It was found that a problem with performing flexible matches is that the matchmaking engine is open to exploitation from advertisements and requests that are too generic. This means that the matchmaking process needs to restrict the return of matches. Only matches that are sufficiently similar to the service request can be accepted. This was achieved with the similarity algorithm implemented in their prototype. It allows a ranking of service matches and allows restricting matches which are below a certain similarity value.

The above related work implements a prototype using application-oriented techniques for semantic matchmaking in Grid environments at the level of services. Grid4All could benefit in both services and resources matchmaking by following a similar approach based on the three selection stages.

A Grid Resource Discovery Method under the Circumstances of Heterogeneous Ontologies

In [7] an approach for semantically discovering grid resources is presented. It concentrates on the ontology heterogeneity problem: when not a common-shared ontology is used for describing the request and advertisement of a grid resource. The approach assumes that Grid resources are classified by a resource ontology and ontology-based queries are expressed on it for selecting the most appropriate resource. An ontology-based query is a boolean expression on the set of concepts names of an ontology. This approach suggests that when different ontologies are used, the queries should be re-written by replacing the names of concepts of the one ontology to other which represents a common basis. The re-writing of the queries is based on a relation matrix that specifies the mapping between the concepts of the ontologies. The mappings are specified based on concept instances: the realization of a set of instances (classification of instances under concepts) to the concepts of the two different ontologies. Hence, common realization of instances under specific concepts implies a mapping between these concepts. Moreover, the queries can be approximated, using upper and lower approximation of concepts, by exploiting the subsumption relations between the concepts in the ontologies. At last but not least, rules are specified for the appropriate re-writing of the queries in order to ensure completeness (must contain all the answers to the original query).

Ontology-Based Resource Matching in the Grid - The Grid Meets the Semantic Web

In [40] a rule-based matchmaking mechanism is presented for selecting computational resources for specific applications. Resources are selected based on application constraints (capabilities of the resource that will be selected) which are specified on request. The matchmaking mechanism consists of 3 components: a) the *ontologies*, b) the *domain background knowledge* and c) the *matchmaking rules*. The ontologies used are: a) a *resource ontology* that provides an abstract model for describing resources and their capabilities, b) a *resource request ontology* that captures requests' properties, e.g. owner of request, characteristics, e.g. the *type of job*, resources' requirements, e.g. minimal physical memory etc. and c) a *policy ontology* that captures resources' authorization and usage policies. The vocabulary defined by the ontologies is used to the creation of the *background knowledge* which constitutes ontologies' A-box. Also, in the *background knowledge* axioms are defined to support reasoning with instances. The *matchmaking rules* define the matchmaking constraints between a request (job description) and resources (resource advertisements). The rules are implemented using the TRIPLE/XSB deductive database system (*TRIPLE*). The requester can specify preference criteria to the list of the result matches.

SPARQL-based OWL-S Service Matchmaking

In [36] a matchmaking mechanism that is based on SPARQL (*SPARQL*)[38] query language is presented. According to this mechanism, a service advertisement is made in OWL-S whereas a service request is formulated using the SPARQL query language exploiting the vocabulary defined by the ontologies used in the service description. Moreover, it is assumed that OWL-S instances of service advertisements are stored in a RDF repository which provides an inferred graph of the instances. The matching is performed in the input and output attributes of the service profile. SPARQL constructs (i.e. optional, filtering and union) allow the construction of complex queries for the selection of services. Also, a scoring function is employed for associating a degree of match in ordering the matching results.

MOD - A Multi-Ontology Discovery System

In [23] a matchmaking mechanism that deals with the case where different ontologies are used for the respective description of service requests and advertisements is proposed. The matchmaking is performed in 4 steps: In the first step, namely *input matching*, the inputs of the advertised services are compared to the inputs of the requested services. In the second step, namely *output matching*, the outputs of the advertised services are compared to the outputs of the requested services. In the third step, namely *operation matching*, service category operation of the requested and service category of the advertised service are matched. In the last step, namely *user-defined matching*, the requester can declare some more constraints or rules to restrict the matching and increase the accuracy of the results of matching. In each step, a concept similarity method, which comes from the ontology mapping research domain, are exploited in order to produce a similarity degree between the concepts involved in the description of the specific service's profile attribute, e.g. input/output attribute etc.

Service Matchmaking and Discovery with Rough Sets

In [51] a matchmaking method that deals with the uncertainty of properties (i.e. properties with empty values) is proposed. In real-world grid settings, it is not always the case that a service advertisement and request uses consistent properties for their description. Usually, service providers and requesters may use their own pre-defined properties to describe services. Moreover, some of these properties might be used in one description and not in other. Most of the matchmaking methods assume that service advertisements and requests use consistent properties to describe services and cannot handle uncertainty on properties. The proposed method, assuming that a common ontology is used for the description of service requests and advertisements, handles the problem of the uncertainty on the attributes of the service profile description producing a list of service matches results.

An Algorithm for Resource Discovery Based on Metadata Semantic Matching in Semantic Grid Environment

In [25] a hybrid matchmaking method for resource selection is presented. The method synthesizes exact and fuzzy matching techniques exploiting the vocabulary defined by the ontologies used to represent resource requests and advertisements. Resource requests and advertisements are made by the population of domain ontologies. These ontologies can be different. Therefore, the method firstly performs matching between the ontologies that have been used for the expression of the request and advertisements and then proceeds to the instances matching (instances represent the specific resource request and advertisement in the corresponding ontologies). The matching is performed on resource characteristics. The degree of matching, in each step of its application, is propagated to the next one resulting in a list of matched resource with various matching degrees.

Resource Matching and a Matchmaking Service for an Intelligent Grid

In [2] providers advertise their resources as instances of concepts of specific Grid resource ontology. On the other hand, a resource request is a tuple of attributes-values pairs and a function that express constraints on the attributes. The names of the attributes are based on the names of the concept's attributes. Hence, the description of resource requests and advertisements is based on a commonly-shared ontology. The function is constructed by attribute names and their specified values along with arithmetic, Boolean and fuzzy operators.

The proposed method returns a ranked list of Grid resources according to their matching degrees. The requester is able to use two matching preferences on this list: a) a matching degree threshold and b) a cardinality threshold. The former one is used for specifying the minimum matching degree of the returned matches whereas the latter one specifies the length of the returned list that contains the matches. Partial match is supported by the method. Although, a resource ontology is used for resources' realization, in general its semantic are not exploited in the matching mechanism but only its structure providing a common vocabulary.

4.3.8 Technology (Ontology languages/reasoners/tools/repositories)

Resource/service description languages & standards

The Semantic Information System will be provided as a service for Web users. This indirectly requires that the subsystems integrated in it must be Web compatible or to perform in an efficient and open way when used in Web applications. The Resources Ontology, the core of the SIS, must be also integrated in SIS and thus it should be compatible with this technological frame.The leading standard for specifying ontologies in the new era of Semantic Web (technologies) is the Web Ontology Language (OWL). As a result to that and to the Grid4All requirements for using the latest technological standards for the modelling of resources/services, we will not examine any alternatives or related technologies. OWL (and OWL-S for services) will be Grid4All technology for representing resources and services. Furthermore, due to the requirement for advanced reasoning facilities and expressivity of the resource description language, OWL-DL (OWL Description Logic) subset or an equivalent-strength language be used. This capability of OWL is also an important reason for choosing it as a language for developing the Resources ontology.

In addition to the selection of the most suitable language for representing Resources/Services, it is important to decide upon the different reasoning engines implementations. Thus, we hereby present a comparison of the most well-known OWL-DL reasoners. There are a variety of criteria for choosing one or another reasoning engine, mainly interesting on their performance when used for placing queries to resources' knowledge bases. These criteria are shown in the following table (first column). Performance and expressivity are important criteria. Querying and reasoning with instances fast is also an important criteria. Conformance with the latest standards such as OWL, JENA repository and SPARQL query language as well as the existence of technical support is criteria that need also to be considered.

	RacerPro[6]	**Fact++**[7]	**KAON2**[8]	**Pellet**[9]
Implementation Language	Java/Lisp API	C++	Java (1.5)	Java
Query Language (QL)	nRQL	**?**	SPARQL	subset of RDQL & RDQL -
QL Expressivity	expressivity of non-recursive datalog with negation	**?**	conjunctive queries, albeit without true non-distinguished variables	conjunctive ABox
Ontology language supported/ implemented DL	SHIQ (OWL-DL without nominal - nominal are approximated-)	sound and complete for SHOIQ (OWL-DL including reasoning about nominal)	SHIQ(D) subset of OWL-DL (all features of OWL-DL apart from nominal)	sound & complete for SHINQ, incomplete for SHOIN with simple datatypes, sound & complete SHOIQ (the expressivity of OWL-DL plus qualified cardinality restrictions)
Interface to Reasoning	DIG/OWL	DIG/LISP	DIG	DIG/OWL/Jena
QL translation	OWL-QL/ SWRL			
Reasoning in T-box (Schema)	subsumption, satisfiability, classification	subsumption, satisfiability, classification	subsumption, satisfiability, classification	subsumption, satisfiability, classification
Reasoning in A-box (instances)	consistency, classification, retrieval: do not support fully the SWRL.	incomplete (retrieval) – no reasoning support for instances	retrieval, conjunctive query answering	retrieval, conjunctive query answering
Weakness	cannot handle nominal & user-defined datatypes, does not support neither default negation nor negation	handle only string and integer datatypes, does not support default negation and function symbols	cannot handle nominal and large numbers in cardinality statements	does not support default negation and function symbols
Specific features	equivalence of roles, synonyms for individuals, datatype property roles, annotation property roles, incremental query answering	synonyms, inverse roles	support SWRL, best A-box reasoning for large KB, KAON2 does not implement the tableaux calculus	Ontology analysis and repair, Species Validation, Entailment, Datatype Reasoning, User-defined Simple Datatypes, Ontology

[6] http://www.racer-systems.com/
[7] http://owl.man.ac.uk/factplusplus/
[8] http://kaon2.semanticweb.org/
[9] http://www.mindswap.org/2003/pellet/index.shtml

		RacerPro[6]	Fact++[7]	KAON2[8]	Pellet[9]
		etc.			Debugging, multi-ontology reasoning. Suitable for S.W. instance heavy applications (support enumerated classes and instance assertions)
License		commercial	open-source (GNU public license)	free (non-commercial use)	open-source
Language Features for Reasoning	Datatypes	OWL-DL datatypes	String, Integers	String, Integers	XML Schema datatypes
	User-derived Datatypes	NO	YES (OWL1.1)	NO	YES (OWL1.1)
	Nominal	NO (approximation)	YES	NO	YES
	Disjunctive classes	YES	YES	YES	YES
	Conjunctive classes	YES	YES	YES	YES
	Inverse properties[10]	YES	YES	YES	YES
	Transitive properties	YES	YES	YES	YES
	Symmetric properties	YES	YES	YES	YES
	Concepts Inheritance	YES	YES	YES	YES
	(arbitrary) Cardinality on Constraints	**?**	**?**	NO	YES
	Qualified Cardinality Restrictions	YES (OWL1.1)	YES (OWL1.1)	NO	YES (OWL1.1)
	Property Inclusion	NO	YES (OWL1.1)	NO	YES (OWL1.1)
	Disjoint properties	NO	YES (OWL1.1)	NO	YES (OWL1.1)
	Anti-symmetric properties	NO	YES (OWL1.1)	NO	YES (OWL1.1)
	Reflexive properties	NO	YES (OWL1.1)	NO	YES (OWL1.1)
	irreflexive properties	NO	YES (OWL1.1)	NO	YES (OWL1.1)
Language Features for Queries	Conjunctive queries	YES	YES	**?**	YES
	Variables in predicate position	**?**	**?**	**?**	NO

[10] Properties are equivalent to roles when we referring to logic languages

		RacerPro[6]	Fact++[7]	KAON2[8]	Pellet[9]
	support for queries with undistinguished variables	**?**	**?**	**?**	YES
	Negation as failure	YES	**?**	**?**	**?**
	KB - arbitrary atomic queries	**?**	**?**	**?**	YES
Performance	Classification	Fast	Fast	?	Acceptable
	Query Answering (conjunctive)	Acceptable	-	?	Fast

Table 3 Reasoning Engines

Storage of Descriptions

Another important aspect concerning Resources/Services descriptions, again related with the performance of retrieving the related knowledge, is the organization of their storage. There are a number of different technologies for storing ontologies, providing more or less efficient mechanisms with respect to scalability, speed, platform, architecture, etc. In the following table we present such technologies for organizing ontologies in database repositories (persistent storage) and/or in-memory organization. The table is organized based on the criteria shown in the first column. The most important criteria are the compatibility with the latest standard (OWL and SPARQL) and the support of reasoning facilities (e.g. Pellet). The performance (speed) of retrieving persistent models (triples) is also of major importance, however such a comparison is not presented in this report due to lack of independed references.

	SESAME 2.0[11]	JENA 2.4[12]	KAON 2[13]	RDF GATEWAY[14]	OWLIM	BigOWLIM
Implementation language	Java	Java	Java	**?**	Java	Java
Status of Latest Version	Alpha	Stable	stable	**?**	stable	beta
Ontology language supported	RDF(S) + other languages via 3rd parties software	RDF (s), OWL	OWL-DL, SWRL, F-Logic	RDFS and OWL	RDF, OWL DLP, OWL HORST, and most of OWL-Lite	RDFS, OWL DLP, OWL Horst
Query language supported	RQL, RDQL, SeRQL	RDQL, SPARQL	SPARQL	RDFQL	SeRQL	RQL, RDQL, SeRQL
Storage (DB) systems supported	In-memory and persistent ORDBMS	In-memory and persistent ORDBMS	persistent ORDBMS	persistent ORDBMS	**?**	**?**

[11] http://www.openrdf.org

[12] http://jena.sourceforge.net/index.html

[13] http://kaon2.semanticweb.org/

[14] http://www.intellidimension.com/default.rsp?topic=/pages/site/products/rdfgateway.rsp

	SESAME 2.0[11]	JENA 2.4[12]	KAON 2[13]	RDF GATEWAY[14]	OWLIM	BigOWLIM
Persistence syntax	RDF, N-Triples	N-Triples			RDF, N-Triples	RDF, N-Triples
API support	YES	YES	YES	YES	Java-YES	Java-YES
Inference supported	BOR reasoner for DAML+OIL and OWL	Jena reasoning for OWL-lite + DIG/API Pellet support for OWL-DL	YES but cannot handle nominals Cannot handle large numbers in cardinality statements	YES	TRRRE engine to perform OWL DLP reasoning (internal reasoning mechanism)	BigTRREE engine
Export support	RDF + other forms via 3rd parties software	RDF, N3, N-triples	**?**	Triples in ASCII	XML, N_Triples, N3	RDF, N-Triples, N3
License (availability)	BSD-style license	Jena license (free use of source and binary)	free precompiled binary distribution	Commercial tools	LGPL open source	on-request

Table 4 Ontology repository systems

4.3.9 Conclusions

Within the context of Grid and the IT industry, many systems addressed the issue of resource representation. Resources are the core of grid, and the way they are described is much important for the grid user to discover and access them. Many related works provide resource description and specification languages together with a matchmaking service that efficiently allocate resources.

A number of recent efforts have focused on Grid-related ontologies. Grid ontologies define fundamental Grid-specific concepts, and the relationships between them. Most of the related works provide a common basis for representing Grid knowledge about Grid systems. Several ontologies proposed so far which are Grid sub-domain specific and have been developed for special purposes. Thus, they can be used for only certain Grid systems/applications (application-oriented resources ontologies).

In respect to Grid economy ontologies, existing Grid ontologies do not contain market specific entities or attributes. Examples of market specific attributes are price, resource availability, etc. Although we have found economic oriented and business ontologies that contain market related attributes none of them seem adequate for representing the trading of resource in the Grid4All context. Thus, Grid4All must develop its own ontology based on (combining) existing efforts of both Resources and Market ontologies/models respectively.

4.4 Distributed Query Allocation for Autonomous Participants: A Survey

4.4.1 Introduction

Increasing numbers of both universities and enterprises are adopting the Internet to share information and do business with others. It already exist several distributed systems that enable heterogeneous participants (consumers and providers) to *(i)* collaborate for reaching a common objective; *(ii)* exchange information; and *(iii)* compete for potential consumers or providers. Providers can be heterogeneous in terms of capacity and data. Heterogeneous capacity means that some providers are more powerful than others are and thus can treat more queries per time unit. Data heterogeneity means that providers provide different data and thus produce different results for a same query.

Besides, in very large-scale distributed systems, participants are usually autonomous, i.e. free to leave the system at any time, and have special interests for some queries. Such interests mainly reflect their *preferences* to allocate and perform queries. Consumers' *preferences* may represent their interests towards providers (e.g. based on reputation)n preferred providers, or quality of service. Providers' *preferences* may represent, for example, their topics of interests, relationships, or strategies.

Example 1. *Consider a provider p_x that represents a courier company. During promotion of its new international shipping service, the provider is more interested in treating queries related to international shipments rather than national ones. Once the advertising campaign is over, the provider's preference may change.*

However, *preferences* are usually considered as private data by consumers and providers (e.g. in an e-commerce scenario, enterprises do not reveal their business strategies). In addition, *preferences* are quite static data, i.e. long-term, while the desire of a provider (resp. a consumer) to perform (allocate) a query may depend on the context and thus is more dynamic, i.e. short-term.

Example 2. *Considering Example 1, p_x may not desire, at sometime, to perform a query related to an international shipment because local reasons, e.g. by overload.*

Thus, consumers and providers are required to express their desire to allocate and perform queries, respectively, via an *intention* notion, which may stem e.g. from combining their *preferences* and other private local consideration such as load (see Figure 36). We can then define the intentions and preferences of consumers and providers as follows.

Preference. A participant's *preference* denotes its topics of interests and some quality parameters that it is looking for (e.g. quality of service and reputation).

Intention. A participant's intention denotes its desire to allocate and perform queries in according to its *preferences* and context (e.g. its load and strategy)

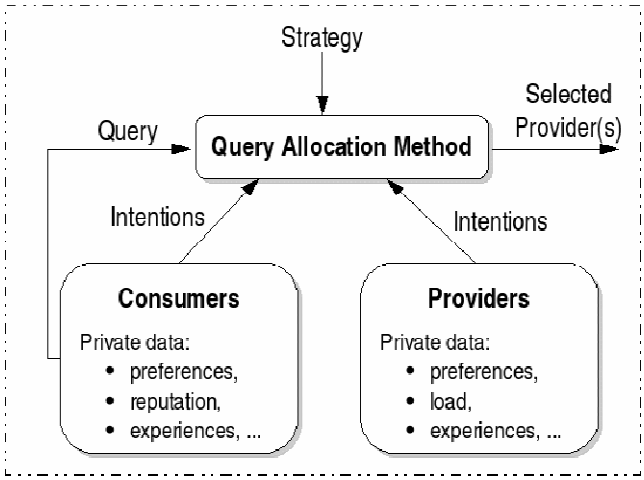


Figure 36 Query Allocation Schema

In these distributed systems, query allocation is a challenge. Participants' autonomy is the main source of the problem, because they may leave the system if they are too dissatisfied. Thus, to survive and succeed in these environments, it is crucial to apply a query allocation strategy that balances queries such that participants are satisfied. In this context, the participant's *satisfaction* means that a query allocation method meets its expectations.

4.4.2 Considering the consumers and participants' intentions

As a motivating example, consider a public e-marketplace where thousands of companies can share information and do business (such as ebay-business [53] and freightquote [54]). Here, business is understood in a very general sense, not necessarily involving money. Each site, which represents a company, preserves its *preferences* for allocating and performing queries. However, *preferences* are usually considered as private data by consumers and providers, e.g. enterprises do not reveal their business strategies. In addition, *preferences* are quite static data, i.e. long-term, while the desire of a provider (resp. a consumer) to perform (allocate) a query may depend on the context and thus is more dynamic, i.e. short-term. For instance, a provider may prefer to perform some kind of queries, but, at some time, it may not desire to perform such queries because local reasons, e.g. by *overload*. Thus, enterprises express their desire to allocate and perform queries via an *intention* notion, which may stem e.g. from combining their *preferences* and other private local consideration such as *load*.

Its is worth remembering that to scale up and be attractive over time, an e-marketplace should *(i)* protect, in the long-run, the consumers and providers' *intentions* for doing business, *(ii)* allow consumers to quickly obtain results, and *(iii)* fairly balance queries so that providers should have the same possibilities for doing business, i.e. to avoid *starvation*, [56].

Consider a simple scenario where a company (*eWine*), which desires to ship wine from France to USA, requests the mediator for companies providing international shipping services, as in freightquote [54]. Here, a query is a call for proposals that providers have to answer in order to provide their services. Consider a second scenario where a company desires to run a specific application, so it requests the mediator for companies providing computing resources (e.g. CPU units), as in [55]. The following details are symmetrical for both scenarios. Suppose that *eWine*, to avoid be flooded by several proposals, desires to receive proposals only from the two best providers that meet its *intentions*, i.e. its expectations. Similarly, to do not waste time in making proposals for uninteresting queries, providers desire to participate only in those negotiations that involve queries meeting their *intentions*.

In these two scenarios, the mediator must perform several tasks. First, it needs to identify the sites that are able to deal with *eWine*'s query (i.e. to find the providers). Next, the mediator should obtain *eWine*'s *intentions* to deal with such providers and the providers' *intention* to deal with *eWine*'s query[15]. One can do so by following the architecture proposed in [70]. Assume that the resulting list contains, for simplicity, only

[15] For simplicity, we assume in this example that the *intentions*' values are binary.

five providers: p_1 … p_5. Table 5 shows these providers with their *intention* to perform the query and *eWine*'s *intention* to deal with each of them. To better illustrate the query allocation problem in these environments, we also show in Table 5 the providers' *available capacity*. However, it is not always possible to know this information since providers may consider it as private. Suppose, then, that p_5 is *overloaded*, i.e. has no more resources for doing business, and that p_2 and p_4 do not intend to deal with *eWine*'s query (notice that this not means they can refuse it) because e.g. p_2 is more interested in its new shipping service to the Asian continent and p_3 has bad experience with *eWine*. Similarly, assume that *eWine* does not intend to deal with p_1 nor p_3 since it does not trust them.

Providers	Providers' Intention	Consumer's Intention	Available Capacity
P_1	Yes	No	0.85
P_2	No	Yes	0.57
P_3	Yes	No	0.22
P_4	No	Yes	0.15
P_5	Yes	Yes	0

Table 5 Providers that are able to deal with the eWine's query

Finally, the mediator needs to select the two most available providers, such that *eWine*'s and providers' *intentions* are respected. Because of enterprises' autonomy, allocating the query to providers that do not desire to deal with the query may cause their departure from the system. Similarly, if the query is allocated to those providers that *eWine* does not desire to deal with might causes the *eWine*'s departure. The only satisfactory option (regarding the consumer and providers' *intention*) is p_5, but allocating the query to it may considerably hurt response time and quality of service as well as the *eWine*'s and p_5's departure. Besides, *eWine* desires to receive two different proposals. So, *what providers should one to select in the above scenarios? Should one consider the consumer's intention? The providers' intention*? *Or providers' available capacity?*

4.4.3 Related Work

We focus on the providers' selection process (the field of distributed databases uses the term query allocation) that just appears as a subproblem of query processing [69]. The providers selection problem is defined as follows. Given an incoming query *q*, the goal of a selection service is to select/rank a set of providers that are able to deal with *q* in accordance to some pre-defined criteria (such as reputation and preferences). The selection of providers is addressed in many domains ranging from distributed database systems to multi-agent systems, P2P data systems, web services, and networking systems. The assumptions and techniques often differ depending on the context and do not result in the same system characteristics. In the following, we draw a picture of the four main related approaches: the multi-agents, web services, load balanced and economical approaches.

Economical approaches

Many solutions [56][61][82] have considered using the principles and models of micro-economy [73] in the field of Computer Science. A survey of economic models for various aspects of distributed system is presented in [61]. In economical models, every consumer tries to minimize its spending to acquire services from the system, and every provider in the system tries to maximize its own profit by selling services to consumers. The motivation of economical models is to decentralize the access to the system's resources that is usually quite complex.

Mariposa [82] pioneered the use of a market approach for dealing with the providers' selection problem in distributed systems. It uses an economical model for allocating queries to providers based on a bidding process. In Mariposa, a *broker* site processes all the incoming queries by requesting providers for *bids*. Then, providers bid for acquiring the query based on a local bulletin board. Finally, the *broker site* selects a set of bids that has an aggregate price and delay under a *budget curve* provided by the consumer. Nevertheless, the mediation procedure is simple and limited. It inherently assumes that consumers are just

interested in low prices and response times. Furthermore, it does not take into account providers' quality and may not process some queries although providers to perform them exist.

Multi-attribute auctions [59][84] are another kind of generalization, which help finding goods providers. The basic idea is that a good is not only qualified by a price but several other attributes like for instance quality. Obviously, in that case quality is attached to an item, while we attach it to the provider in our work. The technical consequence is that price and quality do not evolve the same way at all (for example, in multi-attribute auctions, the price increases if quality increases) leading to different formulas.

In [78], the authors focus on the optimization algorithms for buying and selling query answers, and the negotiation strategy. Their query-trading algorithm runs iteratively, progressively selecting the best execution plan. At each iteration, the consumer sends requests for bids, for a set of queries, and providers reply with offers (*bids*) for dealing with them. Then, the consumer finds the best possible execution plan based on the offers it received. These actions are iterated until either the found execution plan is not better than the plan found in the previous iteration or the set of queries has not been modified (i.e. there is no new subqueries). This approach uses some kind of bargaining between the consumer and the providers, but with different queries at each iteration.

The mechanism proposed by Likhodedov et al [71] aims at maximizing both the consumers and providers' utility. This is difficult because these two goals are contradictory. To cope with this difficulty, the consumer's viewpoint is adopted with a constraint corresponding to the provider's viewpoint. The main principle is the following: the provider puts on sale several units of the same item. The consumer can buy at most one unit each. The provider is not forced to sell all its units, in particular when consumers' proposals are below its reservation price. The provider allocates the units according to its utility (and thus its reservation price). Each consumer must pay the amount that the item would have been worth to it if it had submitted its lowest possible winning bid.

Multi-Agents approaches

In the field of multi-agents systems, the *Contract Net Protocol* (CNP) [81] is often mentioned as a way to allocate tasks. Some agent *A* that wants a task to be completed by another agent sends a call for proposal to its accointances. Those agents which want to complete the task reply by giving the conditions of execution. Then agent *A* compares the offers and chooses the best agent according to its own criteria and informs the agent that has been selected. At first sight, this protocol does not seem very far from an auction protocol. However, the CNP is meant to be used in a cooperative context (i.e. the agents are not self-interested). In addition, it is generally assumed a rather small number of agents, and a detailed description of the conditions of execution, which is not the case in our case.

Several approaches of middle-agents have been defined, [56] [64] [75] [76] and a survey can be found in [68]. These approaches find the providers that are able to treat a given query by matching their capabilities advertisements with the given query. Languages to advertise capabilities have been defined [83]. All these works are efficient but the number of selected providers may remain too large. Recently, some works have investigating the possibility of reducing it by using a notion of quality [77] or word of mouth [85]. The former work clearly suggests to first perform classical matchmaking, and then to refine the obtained selection by considering the providers' quality. The proposal in [85] uses records about each provider, which are obtained using benchmarks and users' feedback. Our model uses a simpler representation of quality, which is just represented as a number. Our proposal strongly differs from these works because the selection process uses not only the providers' quality but also their *bids* for queries, thus allowing them a more active participation in the selection process.

More generally, the notion of quality that we use is related to that of trust and reputation. As for computing reputation or trust, several rather succeeded works exist. They can be found in several works [67][72]. Many works used trust and reputation, in particular in conjunction with other parameters e.g. as in economic models.

The Gorobets et al [66] proposal uses an economical approach to model dynamic systems of interacting agents. Its aims is to ``investigate under what conditions trust can be viable'' in the standard Transaction Cost Economics. In the system, the roles of the agents may be buyers or suppliers. A buyer can make rather than buy if it estimates that it is more profitable for it (there is a tolerance threshold). Transactions occur on the basis of long-run relations between the agents. Each agent establishes a ranking of all its possible alternatives. The agent's scoring takes into account the profit expected from the transaction (through product selling) and trust (expressed as a probability of possible defection of the suppliers). Technically, a parameter enables to support profit more than trust and vice-versa. Each buyer sends a given number of queries to its

most preferred suppliers. Each supplier is free to accept or reject a query, according to its most preferred buyers. If one of its queries rejected, the buyer sends it to less preferred suppliers according to its ordering. This is repeated until the query is accepted or the tolerance threshold is reached. In this latter case, the buyer carries out the query itself. After this matching phase, the selected suppliers produce and deliver for their buyers.

Web services approaches

To locate and select services in the web, web services have to describe properly all their proposed services [56]. Once services have been properly described, these descriptions are made available, via a service directory, to those interested in using them. Service directories allow users to search for and locate services. These directories can be hosted and managed by a trusted entity (centralized approach) or each web service can host and mange them (peer-to-peer approach). Much work on services discovery has been done in the literature.

WSDL attempts to separate services from the concrete data formats and protocols used for implementation. It therefore describes a binding scheme between the abstract service description and its specific implementation. However, WSDL, like UDDI, does not support semantic description of services. DAML-S aims at providing a common ontology of services. It is inspired by other research in the area of the so-called semantic Web that encompasses efforts to populate the Web with content and services having formal semantics. The ultimate goal of this proposal is to provide an ontology that allows users to discover, invoke and compose Web services.

Finally, once located the Web services that provide the desired service of a consumer, the consumer selects the Web service that it wants. The simplest thing is to select the provider with the highest score among the Web services reported by the service directory (the registry). Thus, none of the above works considers the consumers and providers' experiences (feedbacks).

Query load balanced approaches

Much work on providers' selection based on query load has been done in distributed systems [57][63][65][74][79][80][83]. We can classify query load balanced approaches into two categories: those techniques that providers' selection is based on load (*load based*) and those where the selection is based on available capacity (*capacity based*).

Overall, *load based* techniques weigh the providers by load and select a provider with probability that is inversely proportional to its load. Generally, load is defined as the number of queries that providers have in their queue of arrival queries. However, *load based* techniques inherently assume that providers and queries are homogeneous. On the other side, *Capacity based* techniques strive to deal with such heterogeneity by allocating queries to those providers with the greatest available capacity. The provider's capacity is defined as the maximum query rate that the provider can treat.

In contrast to our proposal, all the above works mainly model and address the problem of minimizing the providers' *load* or *utilization* and thus no notion of quality or reputation is considered in the provider selection process.

4.5 Conclusions

This chapter provides a description of the state-of-the-art techniques that are related to the development of the Semantic Information System. SIS in the context of the Grid4All project will implement a service discovery mechanism based on matchmaking between requests and offers of services. It will provide a registry for performing queries in the purpose of discovering available services that fulfil certain criteria imposed by peers within the grid4all environment. These services are:

- Services that expose grid hardware and software resources
- Market services, where grid resources and services are traded
- Services that provide information about other peers that offer or request tradable goods within the Grid4All.

In this context, this chapter describes the functional and the non functional requirements of the Semantic Information System. It describes the relationship of the SIS with other components in the Grid4All, the types of entities (actors) that interact with the system, the functionality that the system exposes to its users and the

characteristics of this functionality. The description focuses on the interaction of the SIS with its users rather than the description of the information that is exchanged during SIS-user interaction. As most of the information that SIS exploits is formalized in an ontology, the chapter outlines the requirements for a resource/service ontology in the frame of Grid4All project, based on economic/market parameters and constraints towards a democratic Grid across society. It then provides the requirements of the technological aspects concerning the implementation of such an ontology, as well as on the matchmaking of resources/services request/offers at the semantic level. Next it provides an extended review of state-of-the-art related approaches on a) resource ontologies and related aspects, b) grid economy and c) semantic matchmaking, and also technologies that support the development, reasoning, querying and storing of ontologies. The chapter thoroughly presents available QoS and policy ontologies. Finally, the state of the art on resources/services matchmaking is described.

The section closes with a survey on distributed query allocation for autonomous participants focusing on the challenge of query allocation. As it is stated, participants' autonomy is the main source of the problem, because they may leave the system if they are too dissatisfied. Thus, to survive and succeed in these environments, it is crucial to apply a query allocation strategy that balances queries such that participants are satisfied. In this context, the participant's satisfaction means that a query allocation method meets its expectations.

References

[1]. Andreozzi S. (2007) GLUE Schema Specification version 1.3, Draft 3 – 16 Jan 2007. http://glueschema.forge.cnaf.infn.it/Spec/V13

[2]. X. Bai, H. Yu, Y. Ji, and D.C. Marinescu, Resource Matching and a Matchmaking Service for an Intelligent Grid, in: *International Journal of Computational Intelligence [IJCI]*, Volume 1, Number 3, Pages 197-205, 2004.

[3]. Beco S., Cantalupo B., Giammarino L., Surridge M., and Matskanis N., OWL-WS: A Workflow Ontology for Dynamic Grid Service Composition, accepted to the *1st IEEE International Conference on e-Science and Grid Computing*, Dec. 5 - 8, 2005.

[4]. Bocchi L., Ciancarini P., Moretti R., Presutti V., Rossi D., An OWL-S Based Approach to Express Grid Services Coordination, In *Proc. 2005 ACM Symposium on Applied Computing*, pp. 1661-1667, 2005.

[5]. Buyya R., Stockinger H., Giddy J., Abramson D. Economic Models for Management of Resources in Peer-to-Peer and Grid Computing. *Proceedings of the SPIE International Conference on Commercial Applications for High-Performance Computing,* 2001.

[6]. Buyya R, Abramson D, and Venugopal. The Grid Economy. *PROCEEDINGS OF THE IEEE*, VOL. 93, NO. 3, MARCH 2005, pp 698- 714.

[7]. T.Chen, B. Zhang, X. Hao, H. Zheng: A Grid Resource Discovery Method under the Circumstances of Heterogeneous Ontologies. In: *Proceedings of the First International Conference on Semantics, Knowledge, and Grid* [SKG 2005], 2005.

[8]. B. Chun, C. Ng, J. Albrecht, D. Parkes, and A. Vahdat. Computational resource exchanges for distributed resource allocation, 2004 (http://citeseer.ist.psu.edu/706369.html)

[9]. S. Colucci, T. D. Noia, E. D. Sciascio, F. Donini, and M. Mongiello. Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. *Electronic Commerce Research and Applications*, 4(3), 2005

[10]. The DARPA Agent Markup Language. http://www.daml.org.

[11]. DAML-S. URL: http://www.daml.org/services/

[12]. Glen Dobson, Russell Lock, Ian Sommerville, Developing an Ontology for QoS, in *Proceedings of the 5th Annual DIRC Research Conference*, pp. 128-132, 2005.

[13]. Erwin D. (2002) UNICORE Plus Final Report - Uniform Interface to Computing Recources. Joint Project Report for the BMBF Project UNICORE Plus. Grant Number: 01 IR 001 A-D, Duration: January 2000 to December 2002. ISBN 3-00-011592-7

[14]. I. Foster et al., The physiology of Grid : An Open Grid Services Architecture for Distributed Systems Integration, *Global Grid Forum*, June 2002.

[15]. Andreas Harth, Yu He, Hongsuda Tangmunarunkit, Stefan Decker, Carl Kesselman. A Semantic Matchmaker Service on the Grid, in *Proceedings of the 13th international World Wide Web conference*, May 17-22, 2004, New York, USA, pp. 326–327.

[16]. Hobbs J. R., Lassila O., Narayanan S. (2001) Towards an Ontology of Resources. http://www.daml.org/2001/09/resources/

[17]. JESS, Java Expert Systems Shell. URL: http://herzberg.ca.sandia.gov/jess/docs/61/index.html

[18]. L. Kagal, T. Finin, and A. Joshi. A policy language for pervasive systems. In *Proceedings of Fourth IEEE International Workshop on Policies for Distributed Systems and Networks*, 2003.

[19]. L. Kagal, T. Finin, and A. Joshi. Declarative Policies for Describing Web Service Capabilities and Constraints, In *Proceedings of the W3C Workshop on Constraints and Capabilities for Web Services*, 2004.

[20]. V. Kolovski, B. Parsia, Y. Katz, J. Hendler. Representing Web Service Policies in OWL-DL, In *Proceedings of International Semantic Web Conference*, 2005.

[21]. KW-fGrid Project: http://www.kwfgrid.eu/

[22]. Lamparter S. and Schnizler B. Trading Services in Ontology-driven Market. SAC'06, April 23-272006, Dijon, France.

[23]. D. N. Le, M.H. Tran, A.E. Soong Goh, MOD - A Multi-Ontology Discovery System, In *Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives* [SMR06], Seoul, 2006.

[24]. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW' 03: Proceedings of the twelfth international conference on World Wide Web*, pages 331–339. ACM Press, 2003

[25]. Zhen Liu, Hongbin Huang, Su Deng, Xueshan Luo, An Algorithm for Resource Discovery Based on Metadata Semantic Matching in Semantic Grid Environment, skg , p. 58, 2005.

[26]. S.A. Ludwig and S.M.S. Reyhiani, Semantic Approach to service discovery in a Grid environment, *Journal of Web Semantics*, pp. 1-13, 2006.

[27]. E. Michael Maximilien and Munindar P. Singh) A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, volume 8, number 5, pages 84-93, September-October 2004.

[28]. METEOR-S project. URL: http://lsdis.cs.uga.edu/projects/meteor-s/

[29]. OGSA: http://www.globus.org/ogsa/

[30]. OWL-S 1.0 rel. 1.0. http://www.daml.org/services/owl-s/1.0/.

[31]. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne1, and Katia Sycara, Semantic Matching of Web Services Capabilities, in I. Horrocks and J. Hendler (Eds.): *ISWC 2002, LNCS 2342*, pp. 333–347, 2002.

[32]. Ioannis V. Papaioannou, Dimitrios T. Tsesmetzis, Ioanna G. Roussaki, Miltiades E. Anagnostou. A QoS Ontology Language for Web-Services. *The IEEE 20th International Conference on Advanced Information Networking and Applications*, Vienna University of Technology, Vienna, Austria

[33]. Parkin M., Burghe S., Corcho O., Snelling D., Brooke J. (2006) The Knowledge of the Grid: An Grid Ontology. Session C1, *6th Cracow Grid Workshop*. 15 - 18 October 2006. Cracow, Poland

[34]. R. Raman, M. Livny, and M. Solomon. Matchmaking: distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 1998

[35]. Kaijun Ren, Nong Xiao, Junqiang Song, Weimin Zhang and Tao Chen, A Semantic-based Meteorology Grid Service Registry, Discovery and Composition Framework, in *Proceedings of the Second International Conference on Semantics*, Knowledge and Grid, 2006, SKG '06, pp. 60–66.

[36]. M.P. Said, A. Matono, I. Kojima, SPARQL-based OWL-S Service Matchmaking, In *Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives* [SMR06], Seoul, 2006.

[37]. Thamarai Selvi Somasundaram, R.A.Balachandar, Vijayakumar Kandasamy, Rajkumar Buyya, Rajagopalan Raman, N.Mohanram and S.Varun. Semantic-based Grid Resource Discovery and its Integration with the Grid Service Broker, *Proceedings of the 14th International Conference on Advanced Computing and Communications* (ADCOM 2006, IEEE Press, Piscataway, New Jersey, USA, ISBN: 1-4244-0715-X, 84-89pp), Dec. 20 - 23, 2006, NITK, Surathkal, Karnataka, India

[38]. SPARQL Query Language for RDF. W3C Home page. URL: http://www.w3.org/TR/rdf-sparql-query/

[39]. K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. Journal of Web Semantics, 1(1), 2003

[40]. H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-Based Resource Matching in the Grid - The Grid Meets the SemanticWeb. In: *Proceedings of SemPGRID '03*, 2003.

[41]. A. Uszok, J.M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken.[2004] Kaos Policy Management for Semantic Web Services. In *IEEE Intelligent Systems*, Vol 19, No. 4 pp. 32-41.

[42]. A. Uszok and J.M. Bradshaw. Kaos policies for web services. In *W3C Workshop on Constraints and Capabilities for Web Services*, 2004.

[43]. Guillermo Vega-Gorgojo, Miguel L. Bote-Lorenzo, Eduardo Gómez-Sánchez, Yannis Dimitriadis, Juan I. Asensio-Pérez. A semantic approach to discovering learning services in grid-based collaborative systems, *Future Generation Computer Systems*, 22 (2006) 709–719.

[44]. Chris Wroe, Robert Stevens, Carole Goble, Angus Roberts, Mark Greenwood. A Suite Of DAML+OIL Ontologies To Describe Bioinformatics Web Services and Data, International Journal of Cooperative Information Systems, 12 (2003), 597–624.

[45]. Web Service Semantics - WSDL-S on W3C, URL: http://www.w3.org/Submission/WSDL-S/

[46]. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1[2]:113–137, 2002.

[47]. WSMO working group. Web service modeling ontology [WSMO] submission, June 2005. W3C member submission. URL : http://www.w3.org/Submission/2005/06/.

[48]. Web Service Resource Framework, http://www.globus.org/wsrf/

[49]. WSRF-S: http://www.wsmo.org/TR/d31/v0.1/

[50]. Xing W., Dikaiakos M., Sakellariou R., Orlando S., Laforenza D. (2005) Design and Development of a Core Grid Ontology. In *CoreGRID Integration Workshop 2005*, pages 21--31, Pisa, Italy

[51]. Bin Yu, Wenming Guo, Maozhen Li, Yong-Hua Song, Peter Hobson, Man Qi, "Service Matchmaking and Discovery with Rough Sets," skg , p. 80, 2006.

[52]. Chen Zhou, Liang-Tien Chia, Bu-Sung Lee, DAML-QoS Ontology forWeb Services, *Proceedings of the IEEE International Conference on Web Services* (ICWS'04) - Vol. 00, pp. 472, 2004

[53]. The eBay sytem. http://business.ebay.com.

[54]. Freightquote. http://www.freightquote.com.

[55]. The Grid4All project. http://grid4all.elibel.tm.fr.

[56]. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture, and Applications*. Springer-Verlag, 2004.

[57]. Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal Computing*, 29(1):180-200, 2000.

[58]. R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. Economic models for management of resources in grid computing. *Computing Research Repository (CoRR)*, cs.DC/0106020, 2001.

[59]. E. David, R. Azoulay-Schwartz, and S. Kraus. Protocols and strategies for automated multi-attribute auctions. *In Procs. of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2002.

[60]. K. Decker, K. Sycara, and M. Williamsom. Middle-agents for the internet. *In Procs. of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[61]. D. Fergurson, Y. Yemini, and C. Nikolaou. Microeconomic algorithms for load balancing in distributed computer systems. *In Procs. of the 8th International Conference on Distributed Computing Systems (ICDCS)*, 1988.

[62]. T. Fong, D. Fowler, and P. Swatman. Success and Failure Factors for Implementing Effective Electronic Markets. *Journal of Electronic Commerce and Business Media*, 8(1), 1998.

[63]. P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. *In Procs. of the 30th International Conference on Very Large Data Bases (VLDB)*, 2004.

[64]. M. Genesereth, A. Keller, and O. Duschka. Infomaster: an information integration system. *In Procs. of the 16th International Conference on Management of Data (SIGMOD)*, 1997.

[65]. Z. Genova and K. Christensen. Challenges in url switching for implementing globally distributed web sites. *In Procs. of the International Workshop on Parallel Processing (ICPP)*, 2000.

[66]. A. Gorobets and B. Nooteboom. Agent based computational model of trust. *Technical Report*, Erasmus Research Institute of Management (ERIM), RSM Erasmus University, 2004.

[67]. M. Kamvar and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. *In Procs. of the 12th International World Wide Web Conference (WWW)*, 2003.

[68]. M. Klush and K. Sycara. Brokering and matchmaking for coordination of agent societies: a survey. *Coordination of internet agents: models, technologies, and applications*, Springer-Verlag, pages 197-224, 2001.

[69]. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422-469, 2000.

[70]. P. Lamarre, S. Cazalens, S. Lemp, and P. Valduriez. A Flexible Mediation Process for Large Distributed Information Systems. *In Procs. of the CoopIS Conf.*, 2004.

[71]. A. Likhodedov and T. Sandhim. Auntion mechanism for optimally trading off revenue and efficiency. *In Procs. of the 4th ACM Conference on Electronic Commerce*, 2003.

[72]. S. Marti. Trust and reputation in peer-to-peer networks. *Phd. Thesis*, Stanford University, 2005.

[73]. A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, June 1995.

[74]. R. Mirchandaney, D. Towsley, and J. Stankovic. Adaptive load sharing in heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 9(4):331-346, 1990.

[75]. M. Nodine, W. Bohrer, and A. Ngu. Semantic brokering over dynamic heterogeneous data sources in infosleuth(tm). *In Procs. of the 15th International Conference on Data Engineering (ICDE)*, 1999.

[76]. M. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, and A. Unruh. Active information gathering in infosleuth. *International Journal of Cooperative Information systems*, 9(1-2).

[77]. C. Ono, S. Nishiyama, K. Kim, B. Paulson, M. Cutkosky, and C. Petrie. Trust-based facilitator: handling word-of-mouth trust for agent-based e-commerce. *Electronic Commerce Research*, 3(3-4):201-220, 2003.

[78]. F. Pentaris and Y. Ioannidis. Query optimization in distributed networks of autonomous database systems. *ACM Transactions on Database Systems (TODS)*, 31(2), 2006.

[79]. E. Rahm and R. Marek. Dynamic multi-resource load balancing in parallel database systems. *In Procs. of the 21 th International Conference on Very Large Data Bases (VLDB)*, 1995.

[80]. N. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *IEEE Computer*, 25(12):33-44, 1992.

[81]. R. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104-1113, 1980.

[82]. M. Stonebraker, P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *VLDB Jounal*, 5(1), 1996.

[83]. K. Sycara, M. Klush, S.Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1):47-53, 1999.

[84]. N. Vulkan and N. Jennings. Efficient mechanisms for the supply of services in multi-agent environments. *In Procs. of the First International Conference on Information and Computation Economies (ICE)*, 1998.

[85]. Z. Zhang and C. Zhang. An improvement to matchmaking algorithms for middle agents. *In Procs. of the First International Joint Conference on Autonomous Agents and Multiagents Systems (AAMAS)*, 2002.

[86]. H. Zhu, T. Yang, Q. Zheng, D. Watson, O. Ibarra, and T. Smith. Adaptice load sharing for clustered digital library servers. *In Procs. of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 1998.

5. Scheduling service – state of art, requirements, characterisation, design

This document provides the first deliverable of the Grid4all Task 2.4. It is produced by the INRIA Grand-Large team. This document aims at providing a state of the art analysis of requirements in terms of scheduling services for classes of applications described in WP4 scenarios.

5.1 Classification of Grids and Desktop Grids

Grid4all aims at building Grid systems deployed within SME or end-users. Typically such systems are composed of very large numbers of PCs, used either within enterprise or run by Internet users. Depending on each environments constraint, such as administrative domain size and professionalism, architecture of these platforms varies. For example, un-trusted and uncontrollable administrative domains may require additional components such as results validation.

Those systems are also made of various software components that interact in order to achieve the desired behaviour. The scheduling service is only one of the components of those systems. So, the design of this component depends on the architecture of the platform.

In this section, we classify Grids and Desktop Grids in different groups according to how resources are distributed. We will start from the Enterprise Desktop Grid that is used in only one administrative domain and end with the Internet Volunteer based Desktop Grid. We will also describe the consequences on the Grid4All scheduling service. We start with a general description of common features found in desktop grids.

Item	Desktop Grid (DG)		Grid
	Internet-based (Volunteer DG)	LAN-based (Enterprise DG)	
Resource	Desktop * Anonymous resource provider	Desktop * within a corporation, university, etc.	Supercomputer, cluster, scientific instruments, database, storage,
Connection	-Non dedicated poor bandwidth -Immediate presence (connectivity) -Consider firewall, NAT, Dynamic address	-Non dedicated intermediate bandwidth -More constant connectivity than volunteer DG	Dedicated high speed, bandwidth
Heterogeneity	High heterogeneous * Need resource grouping	Intermediate heterogeneous * Less heterogeneous than volunteer DG	Low heterogeneous
Dedication	-Non-dedicated -High volatile * Need an incentive mechanism	-Non-dedicated -Low volatile (non-business hours) * Need an incentive mechanism	Dedicated * Be able to use reservation
Trust	Malicious volunteer * Need result certification	Low trustworthy resource provider	High trustworthy resource provider
Failure	Unreliable (faulty)	Unreliable * More reliable than volunteer DG	More reliable than desktop grid
Manageability	Individual-based administration * Totally distributed to individual * Difficult to manage	Individual-based administration * More controllable than volunteer DG	-Domain-based administration * Professional administrator
Application (job)	-Independent (mainly) -Computation-intensive (mainly) -High-throughput (mainly)	-Independent (mainly) -Computation-intensive (mainly) * Data-intensive (possible) -High throughput (mainly)	-Independent or dependent -Computation or data-intensive -High performance (mainly)

Figure 37 A comparison of Grids vs Desktop Grids

5.1.1 General characteristics of Desktop Grids

A Desktop Grid is composed of software and an infrastructure of ordinary PCs which are shared with their owner. The network is a regular IP based network ranging from an enterprise LAN to the Internet. Here we will list features and design goals of those Grids, and then we will describe the general process for running a computation on those systems.

Common features

Desktop grids architects have defined several desirable properties for their systems. Depending on their research topic, some systems' authors choose to emphasize some of the properties in their software. Some of the systems that will be described in this document may not implement all those features. Here is a short list of those properties:

Scalability: Desktop grids must scale to a huge number of nodes. The scale will depend mostly on the number of expected *computing nodes*. An enterprise grid will gather less computing nodes than a volunteer desktop grid.

Heterogeneity: Resources may vary in processor architecture, available memory, operating systems, network connectivity and many other hardware characteristics. Platforms must accommodate this heterogeneity. Manually porting the applications to all available platforms should be avoided.

Connectivity: The system has to provide connectivity for all resources to the desktop grid, even if those resources are behind a firewall or in private address space. Systems may provide mechanisms for storing messages while nodes are off-line.

Volatility: Available nodes and network connectivity vary quickly and incessantly. Network bandwidth and latency also vary. Computation parallelism should adapt to all those platform grows and shrinks.

Fault Tolerance: Increasing the number of involved nodes and, in the case of volunteer computing, using nodes that are not professionally set up, also increase the number of potential faults that may occurs. A failure or loss of one server, client or computing node must have no impact on the correct ending of the computation. Due to the size of desktop grids, faults are very frequent.

Unobstructiveness: Desktop Grid computing nodes are also used by their user; they usually want their own program to run with priority greater than grid applications. Desktop Grid must limit resource usage on computing nodes (including cpu, memory, storage, network usage) Some desktop Grid work by *stealing cycles*: they monitor the node for free resources and run the application when the owner don't use them.

Safety: Internet is well known for not being safe. Servers, clients and computing nodes must be protected from all attacks. Especially, local resources (there applications and data) should be protected from attacks coming from the running application. This is often done by running the software in a virtual machine or in a sandbox.

Correctness: Results obtained from desktop Grid should be validated. The computing node may produce wrong results or its owner may be cheating.

Ease of use: Especially in volunteer desktop Grid, the software should be easy to use in order to be widely deployed.

Performances: Performances of the system should be reasonable.

Anonymity: If needed and accepted by the computing node, sensitive proprietary data shouldn't be usable by the computing node.

Hierarchy: The Grid may exploit the existing infrastructure (network infrastructure, administrative domains).

Rewarding: In desktop Grid, computation resources may be shared by different peoples. Volunteers may want to advertise the amount of resources they offered. (They get credits for the work they compute) Other users may want to charge for their resources or get favour for running there own work when they will need to.

In next sections, we will describe how this general strategy is done in the different types of desktop Grids.

5.1.2 Enterprise Desktop Grids

Enterprise Desktop which consists of volatile hosts within a LAN. LAN's are often found within a corporation or university, and several companies such as Entropia and United Devices have specifically targeted these LAN's as a platform for supporting desktop Grid applications. Enterprise desktop Grids are an attractive platform for large scale computation because the hosts usually have better connectivity with 100Mbps Ethernet for example and have relatively less volatility and heterogeneity than desktop Grids that span the entire Internet. Nevertheless, compared to dedicated clusters, enterprise Grids are volatile and heterogeneous platforms, and so the main challenge is then to develop fault-tolerant, scalable, and efficient scheduling.

First experiments

The first enterprise desktop Grid distributed system is certainly the Worm program [47] at Xerox Palo Alto Research Center. At this time, in the early 80's, each desktop computer involved was mono-application. When they are not used, they reboot, load and execute one "segment" (i.e. job) of the worm which is the distributed application to run. When the computation ends, they reboot to a new worm segment or with their normal software. Applications were simple but of various flavours: distributing news, computing images, alarm clock. Several key ideas that are investigated currently are presented in the paper (self replication,

migration, distributed coordination...). It is interesting to notice that this paper has been partially inspired by a classic science fiction film called The Blob. This experiment involved about 100 machines. During execution, the worm segment avoided disk accesses entirely to limit its obtrusiveness. The worm could grow uncontrollably, and the worm's only control mechanism was a special kill packet that could be broadcasted to kill the worm entirely. In modern-day Internet, some commercial products use a similar mechanism to reboot computers at night for using them for computing without changing their usual working environment.

XtremWeb

XtremWeb [32] [39] [52] is an open source research project at LRI and LAL that belongs to light weight Grid systems. It is primary designed in order to explore scientific issues about Desktop Grid, Global Computing and Peer to Peer distributed systems but have been also used in real computations, especially in physics. First version was released in 2001.

XtremWeb also belongs to the Cycle Stealing Environment family. Various *Activators* are available to decide when cycles can be stolen. (cpu activity, keyboard and mouse, work hours) The platform is written mostly in Java and can run applications written in Java or that use the native platform architecture (versions for Linux, MacOS X and Windows are available) A sandbox is available for isolating the host system from both types of applications.

The architecture (Figure 38) is similar to most well known platforms. It is three-tier architecture with clients, servers and workers. Several instances of those components might be used at the same time. Clients allow platform's users to interact with the platform by submitting stand-alone jobs, retrieving results and managing the platform. Workers are responsible for executing jobs. The server is a coordination service that connects clients and workers. The server accepts tasks from clients, distributes them to workers according to the scheduling policy, provides applications for running them and supervises the execution by detecting worker crash or disconnection. If needed tasks are restarted on other available workers. At the end, it retrieves and stores results before clients download them.

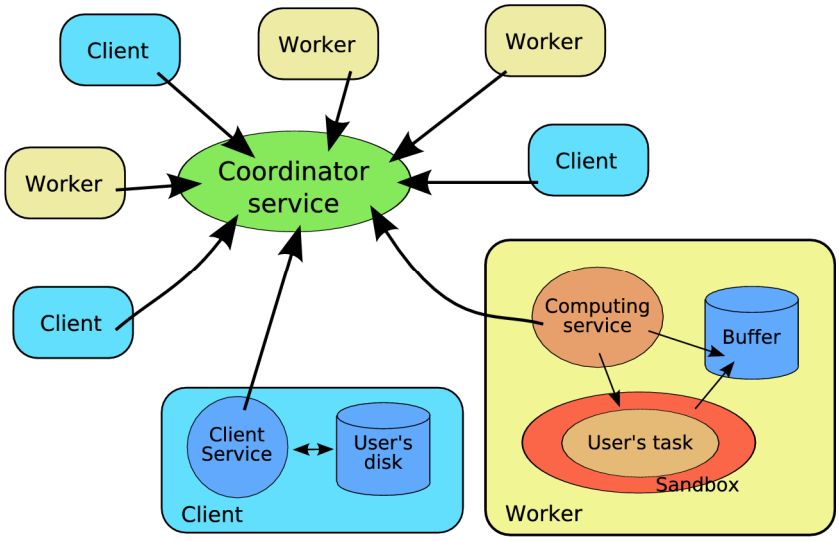


Figure 38 Overview of the XtremWeb platform architecture

Clients and Workers are initiators of all connections to the server which have for consequence that only the server need to be accessible from behind firewalls. Multiples protocols are supported and can be used depending on the type of workload. Communications may also be secured both by encryption and authentication.

Since its first version, XtremWeb have been deployed over networks of common Desktop PCs providing an efficient and cost effective solution for a wide range of application domains: bioinformatics, molecular synthesis, high energy physics, numerical analysis and many more. At the same time, there have been many researches around Xtremweb: XtremWeb-CH, [14] [17] funded by the University of Applied Sciences in Geneva, is an enrichment of XtremWeb in order to better match P2P concepts. Communications are distributed, i.e. direct communications between workers are possible. It provides a distributed scheduler that takes into account the heterogeneity and volatility of workers. There is an automatic detection of the optimal

granularity according to the number of available workers and scheduling tasks. There is also a monitoring tool for visualizing the executions of the applications.

5.1.3 Collaborative Desktop Grids

Collaborative Desktop Grids consists of several Enterprise Desktop Grids which agree to aggregate their resources for a common goal. The OurGrid project [35] [11] is a typical example of such systems. It proposes mechanisms for laboratories to put together their local Desktop Grids. A mechanism allows the local resource managers to construct a P2P network. (Figure 39) These solutions are attractive because utilization of computing power by scientists is usually not constant. When scientists need an extra computing power, this setup allows them to access easily their friend universities resources. In exchange, when their resources are idle, it can be given or rented to others university. This requires a cooperation of the local Desktop Grid systems, usually at the resources managers' level, and mechanisms to schedule several applications. A similar approach has been proposed by the Condor team under the term “flock of condor” [61].

OurGrid has been in production since December 2004 and today aggregates computing resources from about 180 nodes shared by 12 peers. The platform has been limited to supporting bag-of-tasks. Local users have always the priority for their tasks on their local resources, only the unused local resources are shared with other peers. Local jobs kill remote jobs if needed. For promoting cooperation among peers, OurGrid use a *network of favours*. Each peer maintains a matrix of the computing time that it gets granted from other peers. Then, if a processor is requested by more than one peer, it allocates it to the peer with the greatest favor. The favor computation is protected against malicious peers that, for example, would reset its state in order to gain more computing time.

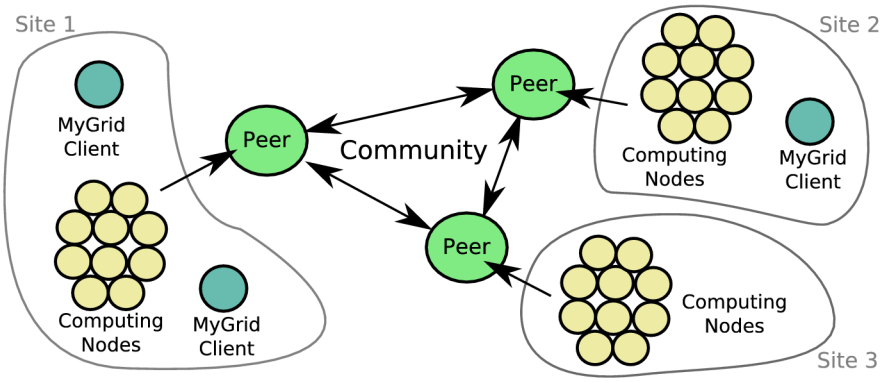


Figure 39 Overview of the OurGrid platform architecture

Other peers are discovered through a centralized discovery system. The network is a free-to-join Grid, so remote peers are not trusted. To address this issue, a sandbox mechanism is proposed (Sand-boxing Without A Name). It is build using Xen. Access to local network is also disabled by this sandbox.

Several scheduling policies have been experimented. The first one, Workqueue with Replication (WQR), was simply sending a random task to the first free processor found. Bad allocation was corrected using replication. In version 2.0 of OurGrid, a new scheduler tries to avoid communication cost by introducing storage affinity. Tasks are sent to computing nodes that are closest to used data. This algorithm tries to avoid the need for redundant information about tasks such as expected completion time. The first algorithm was found to be still more efficient on some cpu-intensive workloads.

5.1.4 Internet Volunteer Desktop Grids

Internet Volunteer Desktop Grids systems have been amongst the largest distributed systems in the world. Projects such as SETI@Home or distributed.net are able to provide hundreds of TFlops on dedicated application from hundred of thousands nodes.

For over a decade, the largest distributed computing platforms in the world have been Internet Volunteer Desktop Grids, which use the idle computing power and free storage of a large set of networked (and often shared) hosts to support large-scale applications. In this case of Grid, owners of resources are end-user

Internet volunteer who provide their personal computer for free. IVDG are an extremely attractive platform because there offer huge computational power at relatively low cost. Currently, many projects, such as SETI@home [22], FOLDING@home [70], and EINSTEIN@home [3], use TeraFlops of computing power of hundreds of thousands of desktop PC's to execute large, high-throughput applications from a variety of scientific domains, including computational biology, astronomy, and physics.

Mono-application Internet Volunteer Desktop Grids

At the beginning of Internet Volunteer Desktop Grid, most of the largest projects were running only one application. Only data were automatically distributed, most of the time using a simple CGI script on a web server. Upgrading the application was requiring that volunteers manually download and install the application. In this section, we will describe some of these projects.

The Great Internet Mersenne Prime Search (GIMPS) [6] is one of the oldest computation using resources provided by volunteer desktop Grid users. It's started in 1996 and still running. The 44th known Mersenne prime have been found in September 2006. Each client connects to a central server (PrimeNet) to get some works. Resources are divided in 3 class based on the processor model and gets different type of tasks. The program only uses 8Mb of RAM, 10Mb of disk space and does very little communications with the server (permanent connection is not required) The program checkpoints every half hour.

Since 1997, Distributed.net [2] tries to solve cryptographic challenges. RC5 and several DES challenges have been solved. The first version of SETI@Home [22] has been released in may 1999. There was already 400 000 pre-registered volunteers. 200 000 clients registered the first week. Between July 2001 and July 2002, the platform computed 221.10^6 workunits at an average rate of 27.36 TeraFLOPS. The programs is doing some treatments on a signal recorded by a radio-telescope and then search for particular artificially made signal in it. The original record is split in workunit both by time (107s long) and by frequency (10 KHz)

The Electric Sheep (http://electricsheep.org/, [37]) screen-saver "realizes the collective dream of sleeping computers". It harnesses the power of idle computers (because they are running the screen-saver) to render, using a genetic algorithm, the fractal animation displayed by itself. The computation uses the volunteers to decide which animation is beautiful and should be improved. This system consists only of one application but, as the project web site claims, about 30000 unique IP addresses contact the server each day and 2Tb are transferred. At article writing time, the unique centralized server was the bottleneck of this system.

BOINC

All these mono-application projects share many common components. So, there was a need for a platform that would provide all these components. Only the part that really does the computation need to be changed for each project.

The Berkeley Open Infrastructure for Network Computing (BOINC) [20] is the biggest volunteer computing platform. More than 900 000 users from nearly all countries participate with more than 1 300 000 computers. [1] More than 40 projects, not including private projects, are available including the popular SETI@Home project.

Each *client* (computing node) is manually attached by the user to one or more *projects* (servers). There is no central server (Figure 40) and most of the scheduling is done by clients. The server is made of several daemons. Each project needs to provide a very small number of different applications but that may be often updated. (Jobs have to be very homogeneous) Each project must last several month mainly because of the time needed to obtain volunteers and their computing resources.

First, *workunits* are produced by a *generator*. Then, the *transitioner*, the daemon that will take care of the different states of the *workunit* life cycle, will replicate (redundancy) the *workunit* in several *results* (instances of *workunits*) Each *result* will be executed on a different client. Then, back to the server, each result will be checked by the *validator* before being stored in the project science database by the *assimilator*.

All communications are done using cgi programs on the project server, so, only port 80 and client to server connections are needed. Each user is rewarded with *credits*, or virtual money, for the count of cycles used on its computer.

The client maintains a cache of *results* to be executed between connections to the Internet. The scheduler tries to enforce many constraints: First, the user may choose to run the applications according to its activity

(screen-saver), working hours, resources available. Second, the user assigns a resource share ratio to the projects. Third, sometimes, some projects may run out of work to distribute.

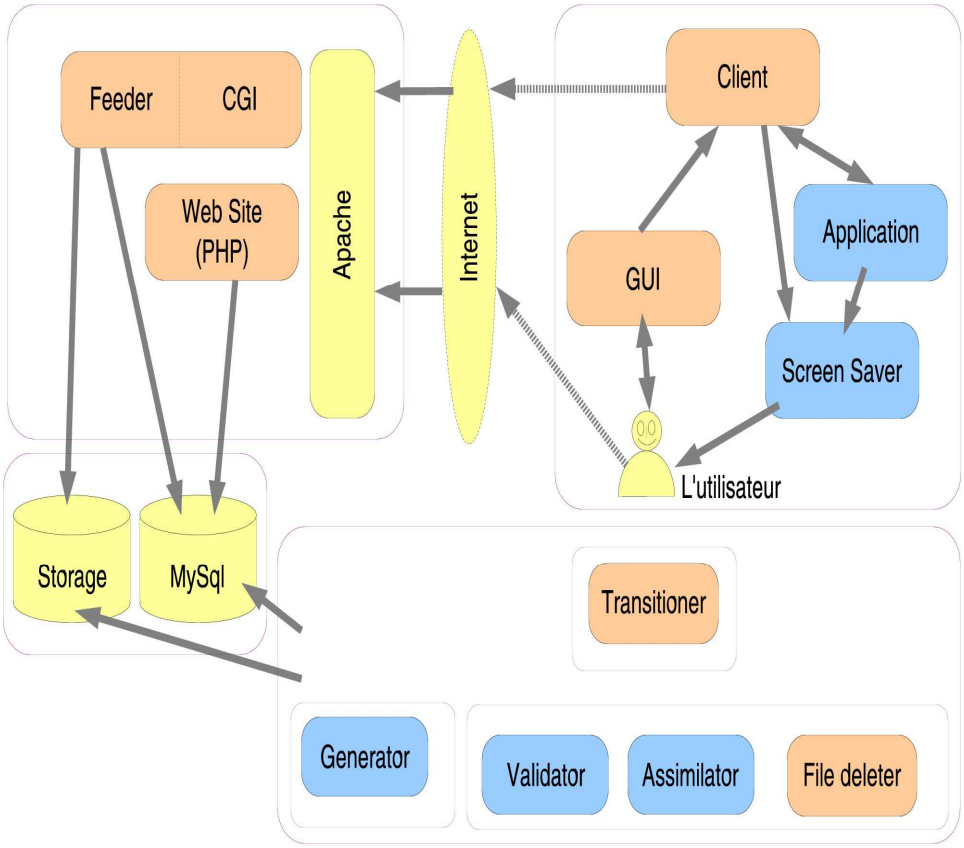


Figure 40 Overview of the BOINC platform architecture

Some others projects were inspired by the BOINC platform. SLINC (Simple Light-weight Infrastructure for Network Computing, `http://slinc.sourceforge.net/`) [25] tries to fix the main limitations of BOINC. They tried to make project creation easier by adding a better documentation. This software is also operating system independent as it runs on the Java platform. It is also database independent (use Hibernate) while BOINC runs only with Mysql. All communications between components are done with XML-RPC and for simplifying the architecture; they have removed the validator component. User's applications are also programming language independent, but only Java and C++ available for now. Two versions of the same application, the first one written in Java, the second one written in C++ have almost the same performance. Some BOINC issues have not been fixed here, such as the time needed to have all the volunteers register their resources.

The POPCORN [57] is a platform for global distributed computing over the Internet. It has been available from mid 1997 till mid 1998. Today, only the papers and documentation are still available. This platform runs on the Java platform and tasks are executed on workers as "computelets", a system similar to usual Java applets. Computelets need only to be instantiated for a task to be distributed. Error and verification process is left to the application level. The platform provides a debugging tool that shows the tree of spawned computelets (for debugging concurrency issues). There is also a market system that enables users to sell their CPU time. The currency works almost the same as BOINC credits. Some applications have been tested on the platform (brute force breaking, genetic algorithm...). At the implementation level, they had some issues with Java immaturity (in 1997-1998).

Bayanihan [69] is another platform for volunteer computing over the Internet. It is written in Java and uses Hord, a package similar to Sun's RMI for communications. Many clients (applet started from a web browser or command line applications) connect to one or more servers.

Korea@Home [24] is a Korean volunteer computing platform. Work management is centralized on one server but since version 2, there is a P2P mechanism that allows direct communication between computing

nodes (*agents*). This platform harness more than 36000 agents, about 300 of them are available at the same time.

5.1.5 Commercial versions of previous Desktops Grids

Enterprises also provide commercial Desktop Grids. Their source code is most of the time unavailable and there is less documentation about their internal components. The server part may be available for use inside an enterprise.

Frontier [5] is a commercial platform launched on June 14, 2000 by Parabon Computation. It can be used for Internet volunteer computing by buying computation power from Parabon. Frontier can be run inside the enterprise (without connected to the Internet) by buying Parabon computing power. Volunteers may be rewarded for the computing power they offer.

First, an application creates a job with different tasks using the Frontier API. Then, the server will assign those tasks to *providers* who are the computing nodes. The task progress may be monitored (with *JobController*) from the server and intermediate results uploaded. When the task is completed, the *Listener* gets the task final result.

From the security point of view, the application must be in Java Virtual Machine bytecode and is executed in a Java sandbox. No communication with other jobs is allowed. The other communications are secured using SSL and clients are authenticated using X.509. The application is protected from the volunteer by obfuscating the Java bytecode and giving no information about the task to the volunteer so it can't guess what is computed on its computer. In the intranet version, some of this security features can be removed: all executables (not only in Java) are allowed and communications between tasks are also allowed.

On this platform, replication is also available. Bad Internet resources are discarded if they fail to compute accurately test tasks, which are sent on a regular basis to *providers*. The *provider* maintains a cache of tasks to run like BOINC does. Tasks may also be prioritized at the server level to get some of the results more rapidly.

There are several other industrial desktop Grid platforms from Entropia [34] (ceased commercial operations in 2004), from United Device [73], Platform [59], Mesh Technologies [9].

System	Organization	Platform	Scale	Resource provider
BOINC	Centralized	Middleware-based	Internet	Volunteer
XtremWeb	Centralized	Middleware-based	Internet	Volunteer
Entropia	Centralized	Middleware-based	LAN or Internet	Enterprise or Volunteer
Bayanihan	Centralized	Web-based or Middleware-based	Internet	Volunteer
Javelin	Centralized	Web-based	Internet	Volunteer
CPM	Centralized	Middleware-based	Internet	Volunteer
Charlotte	Centralized	Web-based	Internet	Volunteer
Popcorn	Centralized	Web-based	Internet	Volunteer
WebCom	Centralized	Web-based	Internet	Volunteer
CCOF	Distributed	Middleware-based	Internet	Volunteer
Organic Grid	Distributed	Middleware-based	Internet	Volunteer
Messor (Anthill)	Distributed	Middleware-based	Internet	Volunteer
Paradropper	Distributed	Middleware-based	Internet	Volunteer
Condor	Centralized	Middleware-based	LAN	Enterprise
Korea@Home	Centralized	Middleware-based	Internet	Volunteer

Figure 41 A survey of Desktop Grid systems

5.1.6 Consequences for Grid4all Scheduling Service

Despite the huge return-on-investment that desktop Grids offer, the platform's use has been limited to mainly task parallel, high-throughput applications. This is a consequence of the resources' volatility and heterogeneity. That is, the resources are volatile in the sense that CPU and host availability fluctuates tremendously over time. This is because the hosts are shared with the owner/user, whose activity is given priority over the desktop Grid application; at any time, an executing desktop Grid task may be pre-empted due to user activity (e.g. key/mouse activity, other user processes, and etc.). Moreover, the hosts are heterogeneous in terms of clock rates, memory sizes, and disk sizes, for example. As a result of such volatility and heterogeneity, broadening the range of desktop Grid applications is a challenging endeavour.

Desktop Grids are composed of a large set of personal computers that belong both to institutions, for instance an enterprise or university, and to individuals. In the former case, these home PCs are volunteered by participants who donate a part of their computing capacities to some public projects. However several key characteristics differentiate DG resources from traditional Grid resources : i) performance; mainstream PCs have no reliable storage and potentially poor communication links, ii) volatility; PCs can join and leave the network at any time and appear with several identities, iii) shared between their users and the desktop Grid applications, iv) spread across administrative domains with a wide variety of security mechanisms ranging from personal routers/firewalls to large- scale PKI infrastructures.

5.2 Classification of DG applications with respect to the Scheduling Service

In this section we present the various classes of application which could be executed within the Grid4all project.

5.2.1 Bag of Task application

Application composed of set of independent tasks is the most common class of application one can execute on a Desktop Grid. This class of applications is straight-forward to schedule and simple to execute when there is few IO. However, it is a very popular class of application, used in many scientific domains. In particular, it permits multi-parametric studies, when one application, typically a simulation code is run against a large set of parameters in order to explore a range of possible solutions.

In [49], several heuristics has been proposed for application which needs a rapid turn around execution. These several heuristics aim at reducing the makespan of the application (the makespan is the time between the submission of the first tasks and the completion of the last task). This is an important feature for a user because minimizing the makespan increases the responsiveness of the platform.

5.2.2 Data Intensive

Enabling Data Grids is one of the fundamental efforts of the computational science community as emphasized by projects such as EGEE [4] and PPDG [60]. This effort is pushed by the new requirements of E-Science. That is, large communities of researchers collaborate to extract knowledge and information from huge amounts of scientific data. This has lead to the emergence of a new class of applications, called *data-intensive* applications which require secure and coordinated access to large datasets, wide-area transfers and broad distribution of TeraBytes of data while keeping track of multiple data replicas. The Data Grid aims at providing such an infrastructure and services to enable data-intensive applications. Despite the attractiveness of Desktop Grid, little work has been done to support data-intensive applications in this context of massively distributed, volatile, shared and heterogeneous resources. Most Desktop Grid systems, like BOINC [21], XtremWeb [40] and OurGrid [23] rely on a centralized architecture for indexing and distributing the data, and thus potentially face issues with scalability and fault-tolerance.

However, we think that the basic blocks can be found in P2P systems. Researchers of DHT's (Distributed Hash Tables) [72, 54, 67] and collaborative data distribution [36, 43, 42], storage over volatile resources [29, 31, 74] and wide-area network storage [26, 51] offer various tools to construct data Grids. To utilize these tools effectively, one needs to bring together these components into a comprehensive framework.

Large data movement across wide-area networks can be costly in terms of performance because bandwidth across the Internet is often limited, variable and unpredictable. Caching data on local workstation storage [46, 58, 74] with adequate scheduling strategies [68, 75] to minimize data transfers can improve overall application execution time. Implementing a simple execution principle like "Owner Compute" still requires the system to efficiently locate data and to provide a model for the cost of moving data. Moreover, accurate modelling [62] and forecasting of P2P communication is still a challenging and open issue, and it will be required before one can efficiently execute more demanding types of applications, such as those that require real-time or stream processing.

5.2.3 Long running (needs checkpoint services)

Long-running applications are challenging due to the volatility of executing nodes. To achieve their execution it requires local or remote check-pointing to avoid loosing their computational state when a failure occurs. In the context of DG these application have to cope with replication and sabotage. An idea proposed in [48] is to compute a signature of checkpoint images and use signature comparison to eliminate diverging execution.

Thus, indexing data with their checksum as is commonly done by P2P software permits basic sabotage tolerance even without retrieving the data.

5.2.4 Real time application

In this paragraph, we focus on enabling soft real-time applications to execute on enterprise desktop Grids; soft real-time applications often have a deadline associated with each task but can afford to miss some of these deadlines. A number of soft real-time applications ranging from information processing of sensor networks [12], real-time video encoding [65], to interactive scientific visualization [53, 71, 16] could potentially utilize desktop Grids. An example of such an application that has soft real-time requirements is on-line parallel tomography [71]. Tomography is the construction of 3-D models from 2-D projections, and it is common in electron microscopy to use tomography to create 3-D images of biological specimens. On-line parallel tomography applications are embarrassingly parallel as each 2-D projection can be decomposed into independent slices that must be distributed to a set of resources for processing. Each slice is on the order of kilobytes or megabytes in size, and there are typically hundreds or thousands of slices per projection, depending on the size of each projection. Ideally, the processing time of a single projection can be done while the user is acquiring the next image from the microscope, which typically takes several minutes [44]. As such, on-line parallel tomography could potentially be executed on desktop Grids if there were effective method for meeting the application's relatively stringent time demands.

While this problem entails a myriad of issues (such as timely data transfers), one needs to achieve probabilistic guarantees on task completion rates via buffering. In [50], we did a work to determine how large a buffer must be allocate to ensure that some fraction of tasks meet their corresponding deadlines. We concentrate particularly on achieving such guarantees on desktop Grids in enterprise environments, for example a company's local area network. This is a challenging because task execution can be delayed or cancelled by users' pre-emptive activity or machine hardware failures.

To enable such soft real-time applications to utilize desktop Grids, we have investigated the use of buffering to ensure task completion rates. In particular, our contribution can be summarized as follows. First, we develop a model of the successful task completion rate as function of the server's buffer size. Second, in the process of verifying the model's assumptions, we show that the aggregate compute power of desktop Grid systems can often be modelled using a normal distribution. Third, in the process of developing the model and running trace-driven simulations, we found several guidelines to be used when scheduling tasks with soft deadlines on volatile resources and describe them in detail.

5.2.5 Wave Scheduling

In [78], the authors describe a time zone aware scheduling method. Tasks are executed in Earth's night zone and are moved from time zone to time zone as the sun rise in the different countries. The authors selected the night because its likely to be a long period of inactivity for the hosts, but, as the article says, another time period may be used.

Applications considered are bag-of-tasks with low communications. Performances are evaluated on simulations but are not trace based. With XtremLab project, an Internet volunteer project running on BOINC, we observed that 41% of hosts are located in the UTC+1 time zone. Some time zone, don't have any host because they are mainly located above seas or inhabited lands. So, in the case of this project, moving tasks from Europe at sunrise may be difficult because others time zone won't have enough resources. Only some of the tasks will be moved.

5.2.6 Network intensive application

There are a few desktop Grid applications that are not CPU or data intensive: they use other resources available on the compute node. The execution time is not limited by processing speed, the amount of available memory, or communication times but by the availability of these resources.

The network is one of these resources. Malicious distributed application (zombies PC) use it for sending a huge amount of data: sending SPAM, distributed attack targeting a given host. But network may also be useful for web spiders. For example, YaCy [15] is a P2P-based search engine. On each volunteer resource, a web crawler collects data from the web that are locally indexed and stored. A local client is available for retrieving search results from other computing nodes through a DHT.

Those tasks often require special scheduling policy from the desktop Grid because usual criteria can't be used. For example, BOINC has support for non CPU-intensive (a special mode that applies to a whole project) tasks but some limitations are imposed: First, the client doesn't maintain a cache of task to run: there is only one task present on the client at a given time. This is due to the fact that BOINC can't estimate completion time by measuring CPU usage as it does for normal projects. Second, non CPU-intensive applications have to restrict there CPU usage to the minimum because there are some other CPU intensive task running at the same time: BOINC doesn't mix scheduling policies.

5.3 Scheduling Service Architectures

How to design a scheduling service depends on several factors: levels of coordination between the scheduling service and the resources, implementation of the scheduling service in terms of a distributed application, integration of the scheduling service within the Grid Environment.

5.3.1 Coordination Requirements

A first criterion to distinguish between the various implementation of scheduling service is the level of coordination between the resources, and the ability for the scheduling service to plan the execution.

As shown in the first chapter, Grid Environments may be more or less coupled distributed systems. In traditional Grid environment, resources are well connected, therefore it is possible for the scheduling service to interact with the resources directly for instance to implement pre-emptive strategy, where the scheduler can interrupt a task running on a computing resources, possibly to give priority to another task.

On the contrary, with Internet Grid, such as the one proposed by Volunteer Computing System are loosely coupled system where resources can provide a very weak level of interaction. For instance, with BOINC, resources connect to a central scheduler to download a set of tasks. Tasks are stored in a local cache by the computing resources which can then go offline and process locally the tasks without maintaining any connection to the scheduler. Thus, the resources ask the scheduler for enough tasks to fill its local cache, so that the resources won't starve during their offline period.

As a consequence, the implementation of the scheduling service, and therefore the set of scheduling strategies strongly depend on the level of coordination reachable by the system. For instance, it would be difficult, if not impossible, to implement a pre-emptive mechanism over Internet Desktop Grids.

Push-based implementation

When the system is strongly coupled a first solution is to use a push-based implementation. In this architecture, developers assume that the resources are always connected to scheduler service. According to this model, tasks are pushed by users to the scheduler, which in turn pushes the tasks to the available resources.

A typical example of this kind of architecture is the MatchMaker mechanism provided by the Condor project. The MatchMaker architecture receives requests from client and computing resources. Its role is to select the best resources according to the client criteria. Once the match is done, client and computing resources directly communicate to exchange tasks and data.

Pull-based implementation

When the system is loosely coupled, composed of a large number of resources subject to frequent failures, a frequent solution is to use pull-based implementation.

With this approach, workers contact the task dispatcher and retrieve tasks from the dispatcher. This solution is by nature more scalable than the push-based implementation because it frees the dispatcher from various tasks such as running a fault detector. However, this approach also suffers from a major drawback: there is no immediate way for the scheduler to control the execution of tasks on the worker. For instance, it might be impossible to verify that a worker is actually running a given task. To provide some control from the scheduler to the worker, DG developers implement a periodic message from the worker to the dispatcher (*trickle* for BOINC, *workAlive* for XtremWeb) which will retrieve special order to control the execution of a task. Overall, push-based implementation allows a tighter control of the application execution compared to pull-based implementation.

5.3.2 Centralized scheduling services

When the number of resources ranges from hundreds to few thousands, a centralized scheduler service can manage the system. Usually the centralized scheduling services will also provide resources reservation if run as a self contained environment. If run in parallel with a Grid system, it will rely on the Grid resource reservation system. Centralized scheduling services are often implemented using 3rd parties software like databases and web servers.

5.3.3 Distributed and coordinated scheduling services

P2P scheduling service

The design of a P2P scheduling service follows the general design of P2P application, which aims at avoiding centralized system. In this case tasks are propagated from node to node using a flooding approach.

Client based distributed scheduling

The multi-project design of the BOINC platform introduces a new scheme in the design of a scheduling service. With this architecture, a worker can participate to several projects. So beyond the scheduler present on the task dispatcher, there is a second scheduler, on the worker side which will decide to which project the worker should retrieve tasks. The client scheduler try to satisfy two type of constraints : i) user's preference expressed in term of participation wishes (I want to contribute 10% of my time to project A, 20% of my time to project B etc) and ii) tasks' deadline, which is a time at a project would like the time to be finished.

5.3.4 Job execution on desktop Grids

We will now describe all the steps that may be needed for executing a job on a desktop Grid. Everything is not done on every platform. The proposed order may vary and some of those steps may happen simultaneously with others.

1. Building the platform: this first step is not trivial because of the number of peoples that may be involved. In volunteer desktop Grid, recruiting volunteers and their resources require some advertising. In collaborative Grid, agreements between institutions may be needed. The owner of the computing node needs to be convinced of the innocuousness of the systems. Then, the software itself needs to be deployed: *clients* (for submitting jobs), *servers*, and *workers*. (for executing jobs) Each component have to find and connect to the other ones, eventually, finding a topology. Granting privileges on the different components may also be needed.
2. Dividing work in sub-tasks: before submitting the work, the application needs to divide it in tasks that will be executed on different computing nodes. The parallelism is created here.

3. Submitting the application: On some systems, the application is submitted separately and before data to compute. Users may need to submit multiple versions of the same application for running on the different platforms that will be used.
4. Submitting tasks and data: Then all the tasks are submitted to the system. The system checks that the user is allowed to submit tasks.
5. Fault tolerance, correctness: Depending on the used algorithm, tasks may be replicated or restarted here to tolerate faults and discover errors.
6. Finding resources: Then the system will find resources for running all the tasks. In many systems, free resources will contact the server for getting work. In other systems, the server will distribute the tasks. This global meta-scheduling must take care of available resources on computing nodes. (CPU, memory, free disk space, software license)
7. Sending tasks to selected resources: The tasks and needed input files are transferred to computing nodes. The method used here depends on the amount of data to transfer.
8. Caching tasks on workers: because of network availability, the computing node may maintain a cache of work to be done.
9. Local scheduling of tasks: a local scheduler checks if conditions are meet for stealing cycles. Tasks that won't be finished before the deadline may be aborted. If there is a cache of work, the scheduler will decide which to execute first.
10. Running tasks: Tasks are executed by their application. The system enforces resource limitation and security. Communications may be possible with other computing nodes but should not be possible with local network. Tasks and computing nodes are monitored.
11. Sending results back: Results are send back to the server. Eventually error conditions are reported.
12. Validating results: Results are validated to ensure there correctness.
13. Rewarding worker: If the reported result is correct, the owner of the corresponding computing resource get points or privileges for running its own tasks.
14. Reporting tasks completion or failure to the client.
15. Retrieving results: depending on their size and the system, results may be stored on the computing node or on a result server. The system must provide a way to retrieve those results.
16. Deleting old data: old and unused results, input data and application need to be removed. The desktop Grid can't be easily reset and usually last for a long time, so, a cleaning mechanism must be provided.
17. Removing resources: All the resources of the desktop Grid may leave the system at any time. The system may not always be advertised of this event. Tasks assigned to such a computing node need to be reassigned to another node.

5.3.5 Implication for the Grid4all scheduling service

Traditional scheduling service for desktop grids is enclosed in a self contained environment. For the Grid4all project, the scheduling service will have to cooperate with other services such as the resource allocation, resource monitor and market service. It will be implemented as an independent service using the Grid4all middleware envelope.

5.4 Overview of scheduling heuristics

A scheduling problem is defined by an application model and platform model, and a schedule is defined by when and where tasks of the application should utilize resources in the model. Often, these models differ by the amount of information known *a priori* by the scheduling algorithm, and the dynamism of the different parameters of the models. Some relevant parameters of the application model include dependencies among tasks, the size of tasks and whether it can be mouldable, and whether this information is known *a priori*, whether the arrival rates of tasks are periodic or not, whether tasks can be pre-empted, and the computation to communication ratio of each task. Some relevant parameters of platforms of the platform model include the number of ports of the server usable for distributing tasks, resource and interconnect speeds and

topologies, and the volatility and heterogeneity of resources. In the following sections, we describe two of the most relevant scheduling research areas with respect to the scheduling service, detailing the application and platform models in each respective area.

5.4.1 Deadline Scheduling

In deadline scheduling, the goal is to meet a set of deadlines for a group of tasks. Feasible solutions have been obtained for only a few simple cases. For example, for the problem where not all task characteristics are known from the start and when tasks arrive dynamically, the authors in [18, 55] show that the optimal algorithm is earliest deadline first (EDF) on a single processor machine with independent and pre-emptable tasks. The authors in [18, 55] show that the least laxity (LLF) algorithm is also optimal. That is, if a feasibly optimal schedule exists, EDF or LLF are guaranteed to find it.

However, with the exception of a few simple cases, many versions of the deadline scheduling problem do not have optimal algorithm. For example, with dynamic scenarios with more than a single processor, the authors [18, 55, 56] in show that an optimal algorithm does not exist without *a priori* knowledge of task deadlines, computation times and arrival times using a adversarial approach for their proofs.

5.4.2 Divisible Load Scheduling

Application Model

A divisible workload application consists of a computation that can be divided into arbitrarily smaller independent computations often called *chunks*. Traditionally, divisible workload applications are relatively data-intensive with relatively low computation to communication ratios. Many scientific and domestic applications fit this description, such as MPEG encoding [8], photo batch processing [41, 10, 13], and biological sequence alignment and volume rendering.

In [63], the authors characterize several real divisible workload applications (see Table 1). They characterize HMMER [7], which is a sequence classification algorithm from computational biology. Then they characterize a MPEG4 video compression application [8]. Finally, they characterize a volume rendering application called VFLEET [13].

For each application, they describe the input size in MB, the running time in seconds on an Athlon 1.8GHz machine, and the computation to communication ratio R assuming a 100 Mb/s data transfer rate (see Table 1). They find that these applications differ in their defining characteristics by as much as an order of magnitude.

Further the authors show the coefficient of variation (which is the standard deviation divided by the mean) of the amount of computation per unit of load expressed as a percentage (labeled as G in Table 1). Finally, the authors compute the range of running times over all units of load. Because of data-dependencies or non-deterministic computation [63], the coefficient of variation can be as high as 10%, and the spread can be as high as 2700%.

Application	Input size (MB)	Running time (sec)	R	G	(max - min) / mean
HMMER	802.0	534	6.7	9%	2700%
MPEG	716.8	2494	34.8	10%	30%
VFleet	87.5	600	68.0	1%	2%

Table 1: Characteristics of several applications: input size, running time on a 1.8GHz Athlon, computation to communication ratio (R), coefficient of variation of a unit of load (G), and spread of running time of unit of load. [2]

MPEG4 video compression for a particular application has an input size of 716.8 (MB) and running time of 2494 seconds. Assuming a 100 Mb/s connection, this gives a computation to communication ratio of 34.8

with a coefficient of variance of 10% due to data-dependent and/or non-deterministic computation. Note that with the recent advent of Gigabit networks at homes, the computation to communication ratio could become higher.

Platform model

Most of the divisible workload algorithms assume that there is a server from which the divisible workload application is distributed among workers. Moreover, the server has only a single port, so only 1 datum can be sent to a worker at a time. Also, most algorithms do not consider variability in the CPU availability of the workers. That is, they assume the machines are dedicated for scheduling purposes. Only recently in [63] has the issue of variable availability been examined. Also, only recently has the network overhead (in particular latency) of sending the data been addressed using simple affine functions to model the initial overhead of sending the data.

Scheduling Algorithms

The goal in scheduling a divisible workload algorithm is to minimize the applications makespan, which is essentially its completion time. To understand the scheduling issues with divisible workload algorithms, it is useful to examine issues with the most naïve algorithm. That is, a naïve approach to send a divisible workload algorithm would be to divide the workload into a number of equally sized tasks, and then to distribute these tasks at the same time to the workers. The main problem with the naïve approach is that while the workers are waiting for the data to arrive, they could have been doing computation. Instead, they wait until all the data has been transferred before starting to compute.

This issue of overlapping of data and computation is what divisible workload scheduling (DLS) algorithms address [63, 77, 27, 28, 38, 66, 19, 45]. At a high level, the algorithms overlap data transfers with computation by sending a small data chuck to a worker so that it can immediately begin computation, and while it the worker computes, the server sends another chunk, and so on (see Figure 42).

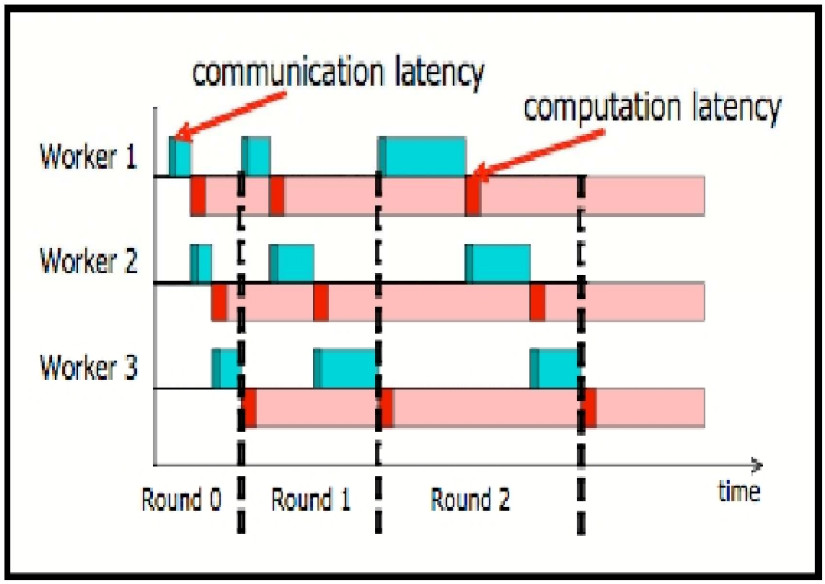


Figure 42 An example of a divisible workload schedule [33]

Notice in Figure 42 where there are several rounds in which data is distributed to workers. Distributing data in rounds can have several benefits. First, the initial idle time of resources in Round 0 is lessened when using multiple rounds. Second, using multiple rounds near the end of the application can be a means of dealing with variable host load. That is, by having relatively smaller rounds near the end of the application, the algorithm can detect an imbalance of distributed work more quickly and then it can act responsively when fluctuations occur.

Divisible workload algorithms can be categorized broadly using two criteria. The first criterion is how they form the rounds (for example, increasing or decreasing chunk size with each round) and the number of rounds (for example, using only a single round versus multiple rounds). The second criteria is how they

model the costs of data transfers to the resources (for example, assuming the time to transfer data is proportional, i.e., linear to its data size, versus taking into account start-up costs using affine functions).

One of the most effective for divisible load applications on heterogeneous and shared platforms is called Robust Scheduling for Divisible Workloads (RUMR) [76]. RUMR is able to compute a near-optimal number of rounds while taking into account the cost of communication and computation using affine functions. Moreover, RUMR is amenable to uncertain communication and computation times as it decreases the size of chunks used towards the end of the application.

Software for deployment

Despite over a decade of algorithmic research with divisible workloads, developers implemented real systems for divisible workload applications only recently. One of the most relevant systems is APST-DV [63] which is a user-level tool for deploying divisible workload application across a set of distributed and potentially shared resources. APST-DV provides a simple XML interface by which a user can specify his/her divisible workload application and implements a number of divisible workload algorithms. In the XML input, the user indicates the executable, the input file name, and its divisibility. The divisibility of the input file consists of the method by which to divide the input (for example, uniformly or by a user-supplied call-back function, the unit of data, and the scheduling algorithm).

Then APST-DV takes the inputs and deploys a scheduling specified by the algorithm. To deploy the schedule, APST-DV can control or communicate with resources using a number of different mechanisms. With respect to controlling task execution, it can use fork, GRAM, NetSolve, ssh, Condor, PBS or LoadLeveler. With respect to communication, it can use cp, scp, FTP, GASS, GridFTp, SRB. Finally to monitor the state of the system (for example, host load), which is simulation for precise scheduling, it can use a number of services, such as MDS, NWS, or Ganglia. As setting up these services might not be trivial, an alternative approach is to use the application itself (or the small part of it) as an active probe to estimate the performance of the applications. This is the approach that the authors take in [63].

In terms of real experiments with the APST-DV software, in [63] , the authors execute the encoding of a 716.9MB video file over 7 hosts (one 700MHz AMD Athlon, and six 1.73 AMD Athlon XP's) connected by 100MB/sec Ethernet. Over 10 runs with each scheduling algorithm, the execution time for each run ranged between 12 and 21 minutes. (On a single machine the time to encode the video was 41.6 minutes.)

5.5 Grid4All Scheduling Service

In this section, we describe the different services (and the specific API's) provided (or not provided) by the VO management and scheduling entities. Below we describe the entities and roles involved with job execution and a job's lifecycle.

5.5.1 Applications

Photograph Transformation and Video Compressions

The applications of domestic users targeted by the scheduling service are photograph batch processing and video encoding applications. We envision users submitting batches of photo for filtering (for example, de-blurring, sharpening or noise reduction), which often requires significant computational processing of each pixel and its neighbours.

One particular algorithm we will use is the photograph de-blurring algorithm described in [41]. Blurry photos are a consequence of camera motion during exposure. Longer exposures are often required in dim light settings, for example indoors or at night, can increase the chance of blurriness due to camera shake.

The de-blurring algorithm described in [41] can be used to reduce blurriness in digital photos that occur as a result of camera shake. The camera blurring process can be modelled as a convolution, i.e., an operator that works on two functions (f and g) to produce a third (h). In terms of photography, f is the original clear picture,

g is the signal convolved around f, and h is the blurry photograph. To de-blur a photograph, one needs to perform de-convolution, which involves the estimation of the unknown g using statistical estimation techniques.

The algorithm itself is divided into three stages, namely pre-processing, kernel estimation, and image reconstruction. The most important and time-consuming step is kernel estimation, where the algorithm estimates the blur kernel from the input data. To achieve this estimate, the user (or an agent on behalf of the user) must provide the original blurred image, a rectangular patch within the image, an upper bound on the amount of blurriness specified in pixels, and an initial estimate of the direction of the blurriness (horizontal or vertical).

The quality of the de-blurred result of algorithm improves with the size of the rectangular patch from which the blur kernel in inferred. However, at the same, the running time of the algorithm is dependent on the size of the patch. Thus, with a single resource, one cannot afford to use the whole image as the patch, especially if there are multiple photographs to process.

The Grid4All distributed batch processing service will allow one to distribute the de-blurring computation on multiple resources, to improve the quality and speed of the de-blurring process. Currently the code is written in MATLAB and copyrighted by the Massachusetts Institute of Technology. However, we are looking into options of a license-free compiled version of the code.

In addition to photograph batch processing, we envision users submitting videos files for encoding, i.e., compression to MPEG formats. A video file consists of a set of frames. Encoding algorithms compress information within individual frames and among a series of frames. While DV encoded files may be divided in frames, frames carry different of amounts of information and thus may be of different sizes. Moreover, the amount of information that may be compressed differs from frame to frame. Thus, the actual chunks of the inputs files may not be of identical sizes, and the running time for compression may differ for any subset of frames (including subsets of size 1). Divisible workload algorithms that assume arbitrarily divisible workloads and uneven computational running times of the non-uniform chunks pose an interesting challenge, which we will investigate.

The computation to communication ratios for these real photo and video applications are relatively low, indicating that these applications are data-intensive. This presents several challenges in terms of scheduling data transfers and computation, as was discussed previously.

Collaborative Network Simulation Environment

Network simulation is an integral part of both network education and research. In particular, packet-level network simulation, as described in detail by work package 4, is a compute and data-intensive task that often results in large outputs. While network simulation can be seen as a workflow (consisting of processing, post-processing, and visualization steps), these steps from the perspective of the scheduling service equate to a single task. That is, we will assume at least initially that the steps in running a network simulation can all be done on a single host. Thus, in a collaborative educational environment where many students might be simultaneously running network simulation experiments, each simulation experiment is seen as a single task, and the scheduling service has to simply schedule a set of independent tasks. As such, this scheduling scenario is subsumed by the more complex scheduling scenario of scheduling data-intensive divisible workloads, and the functionality of the scheduling service for divisible workloads can be reused for enable network simulation in Grid4All as well.

5.5.2 Entities and Roles

Below we describe the entities and their roles involved with scheduling and resource management. We also list functions in the API to be exported by each entity.

User agent

This is the entity that specifies a job. It could be implemented through a portal, i.e., simple web interface to submit, monitor, and control jobs.

Scheduling Service

The scheduling service determines a mapping of tasks to resources within a VO. Specifically, it determines when and where (that is, between which resources) input or output data transfers should occur, and when and where tasks should be executed to meet the job deadline (or be within some range of the deadline as defined by the laxity). For each set of resources, the scheduling service will conduct a simulation of a divisible workload algorithm in real-time to determine whether or not the deadline can be met. All feasible schedules are then returned to the caller.

API: <schedule_1…schedule_n> find_schedules (<resource_set_1>, <resource_set_2>… <resource_set_n>, <job>)

A `<resource_set>` consists of a set of tuples, one for each resource, each of which contains the following information: `<resource_id, start_time, duration>`. A `<job>` consists of `<data_size, step_type, step_size, deadline, laxity, src_resource_id>`. Note that because the data file (for example, a DV encoded input file) itself could be relatively large, we only pass a description of it to the scheduler. `<data_size>` is the total size of the input file. `Step_type` is the unit of the `step_size` (for example, bytes). `<step_size>` is the desired chunk size of the input file to be divided. (The data itself may or may not be divisible into uniform chunks, but `<step_size>` is meant to be a first order approximation.) The deadline is a relative value that is with respect the initial start time of the job. The location of the submission node can be determined using the `src_resource_id` in the resource information service. `<schedule_x>` are feasible schedules where the deadline can actually be met.

Autonomic resource management framework

As presented in Chapter 1, the task T2.1 of this work package will provide several services relevant to the management of middleware across a large-scale distributed platform [30] and in particular for management of virtual organisations. This architecture is based on a control loop that uses sensors (providing monitoring hooks to managed elements), deciders (which determine what action must be taken), and actuators (which perform the determined actions). The management components stores and maintain information about the system composition (e.g. list of nodes, which nodes are available or unavailable, operating system, software modules), the system architecture (e.g. relationship between components, which nodes contained which software components), and the system state (e.g. whether the node is active or failed). An important service provided by T2.1 is that of service deployment. Components and their bindings specified via the Architecture Description Language (ADL) are deployed through the deployment management service.

The scheduling service uses this framework by providing the name of the component (in this case, the Java class that contains the wrapper for APST-DV), configuration values (for example, the APST-DV worker daemon configuration file indicated through get and set properties), and the interface of the wrapper (that is, the command line required to start and stop the worker daemons).

API: deploy (<ADL>)

Buyer Agent

Resources to schedule applications are acquired through a buyer agent that executes within the scope of a virtual organisation. This agent uses the Grid resource market (described in Chapter 2) to discover and acquire resources for a VO. Specifically, the buyer agent is doted with a budget and its goal is to find the cheapest set of affordable resources from the various markets, on which the user's job can run. The notion "can run" is defined later. The buyer agent interacts with the Semantic Information Service (Chapter 3) to lookup available running markets, to determine various estimators of resource prices (the average, maximum

and minimum price of resources within each market), and to determine the price on a set of resources. Given a budget, the buyer agent can determine the number and types of resources that can be afforded.

API: <resource_set_1>… <resource_set_n> get_resources_sets (<job>, <budget>)
<market> find_market (), register_market (<market>)

The buyer agent is used by the scheduling service in two steps. In the first step, the BA is requested to obtain possible resource sets that may be acquired at the markets within the limits of the budget. In the second step the BA acquires (tries to) one of the resource sets.

Resource information service

The VO management system will support basic queries (e.g. from the scheduling service) for the description of resources that it manages. The types of supportable queries include the hardware characteristics and location (e.g. city) of the resource.

API: <CPU, memory, max_network_bandwidth, city> get_charac (<resource_id>)

5.5.3 Job lifecycle (and the interaction between entities)

Overview

The main steps of a compute job's lifecycles can be summarized as follows:

1. **Job submission** – A job with information about the inputs data sizes and how it can be divided is submitted to the buyer agent along with the user's budget.
2. **Resource discovery** – The buyer agent determines several sets of affordable resources using coarse and static information about the resources (e.g. location and clock rates).
3. **Resource selection and scheduling** – Given each set of resources and job information, the scheduling service determines which set of resources yield feasible schedulers using real-time simulation.
4. **Resource binding** – Once the resources in a feasible schedule have been bought by the buyer agent, the scheduling service deploys the schedule by specifying the ADL to the Jade management framework.

Below we describe the lifecycle of a job in detail by first listing our assumptions, and then the key steps involving resource management and scheduling required before execution of the schedule. In an effort to make things more concrete, we list the corresponding function call(s) exported by each service that are associated with each step.

Assumptions

Resource reservations are not allowed as it would require a system for cancelling reservations, and trying to add resource reservations would make resource management prohibitively complex.

Once a lease for a resource is purchased, we assume that the lease will not be intentionally broken. Of course, this does not preclude leases broken as a result of hard failures. Also, once leased, we assume the resource will not be shared with other user processes. In scheduling terms, this means that tasks may not be pre-empted. We assume only a bid-clear system (not bid-decide) and that the buyer agent cannot be expected to provide subnet information.

Job submission

User agent submits <job>, <budget> to buyer agent. A <job> consists of <data_size, step_type, step_size, deadline, laxity, src_resource_id>.

Resource Discovery

The idea is to use coarse and static information about the resources and the fact that the video and photo processing applications are relatively data-intensive, to simplify initial resource discovery and to conduct the "bootstrapping" required to determine initial sets of resources to be considered by the scheduling service. We present below one possible way to determine sets of affordable resources:

Given the compute node on which the job (including its input data) initially resides (this can be determined by submitting a query to the resource information service with the `src_resource_id`), the buyer agent determines a set of resources within the same city (call it the "close-by" set), and a set of resources in a neighbouring city (call it "far-away"). The point is just to use proximity to greatly reduce the valid search space for doing resource selection, that is, the buyer agent is asked to search for resources that are close to the compute node on which the input data resides.

Given a job, different possible resource sets may enable the job to terminate within its deadline. These different resource sets are initially not known. A compute node is characterized by its processing capacity, by its distance to the node where the input data resides. The number (or total processing capacity that may be acquired) of nodes that may be acquired depends on the current market prices and finally the times at which the resources may be acquired also depends on the current market demand for the resources. We proceed by heuristics to narrow down the configurations that the buyer agent should seek to acquire (subject to budget and current market prices).

Assuming that resources need to be acquired at any time from current time, the possible resource sets are classified as following:

- <close-by, short duration>
- <close-by, long duration>
- <far-away, short>
- <far-away, long duration>

The buyer agent chooses two initial starting values (short [.5**deadline*], long [*deadline*]). For each set of resources (close-by and far-away), and for each duration (short, long), the buyer agent ranks the resources in terms of price (using e.g. average resource price per market as provided by the market information service).

For each ranked set the buyer agent forms two maximal subsets that are within the budget constraint, namely a subset of "cheap" resources (start from the cheapest and going up the ranking), and a subset of "expensive" (starting from the most expensive and going down). Adding the price dimension results in eight possible configurations, that is, <close-by, short, cheap>, <close-by, long, cheap>, <close-by, short, expensive>, <close-by, long, expensive>, <far-away, short, cheap>, <far-away, long, cheap>, <far-away, short, expensive>, <far-away, long, expensive>.

The buyer agent determines the configuration that may be acquired within its budget by querying the market information services (Chapter 9). It of course cannot know whether the job's deadline can be met given each set of resources. The scheduling service is asked to estimate the job termination time for each of the resource sets that may be acquired currently at the market (obtained through the API get_resource_sets).

Resource selection and scheduling

Each set of resources, and the job is given to the scheduling service. The scheduling service can be wrapped as a Fractal component to provide a standard management interface that will allow itself to be managed by Jade as an internal component. Web services will be used for external components. The service tries to schedule tasks among each set of resources, determining a completion time and whether the deadline can be met. It returns a schedule (that is, which resources to use and when for computation and network transfers) to the buyer agent for each resource set, including an estimated completion time.

The buyer agent proposes each estimated completion time and price (which may have changed depending on market price fluctuations and how long the scheduling service took to return) to the user, who decides which schedule if any should be executed. However, the most time consuming step is probably the scheduling simulations. So, if any change in pricing for a set of resources occurs, it would most likely be after

this process. If however none of the schedules are acceptable then the bid is cleared, i.e., nullified. In later versions of the system, we make allow for a bid-decide-revise-bid loop.

<schedule_1 … schedule_n> find_schedules (<resource_set_1> … <resource_set_n>, job)

Resource Binding

Once a schedule is approved by the user agent, the buyer agent purchases the resources, which form a VO, and then the schedule is executed, which requires binding. There are two levels of binding. First, the middleware (most likely APST-DV) must be installed and deployed on the set of the resources within the VO. The deployment service presented in Chapter 1 will be used to install, deploy, and manage the middleware components on the set of resources, specifying the submission node as the server since that is where the input data resides.

deploy (<ADL>)

Second, the set of tasks in the job must be distributed to the set of assigned resources. The middleware will ensure that the tasks are distributed to the set of resource according to the decided schedule.

Middleware::job_submit (<job>, <schedule>)

5.5.4 Implementation

In the previous sections, we described the specification for the scheduling service and the components it interacts with, namely the resource management system, and the deployment system.

We are considering using APST-DV as a means of deploying the actual schedule. APST-DV is licensed by the University of California and allows for research and non-profit uses of the software without fees or written agreements. To address licensing concerns, we will integrate APST-DV with the scheduling service framework using wrappers and in a modular fashion. This way, APST-DV could be easily replaced in the future by other software that offers similar functionality but under a less restrictive license.

5.5.5 Future Challenges

There are several open issues that will be addressed. First, it is not clear whether a constant <step_size> used for the real-time simulation will in fact give an accurate estimate for the real execution given that the step_size for the real application is often variable from photo to photo or frame to frame.

Second, we must be careful that the simulation time does not take too long. Otherwise, we risk large changes in the pricing of the resources or that the resources get bought by a competing party (as no reservation mechanism is available.)

Third, we must ensure that the estimated execution time matches the real execution. This will require accurate models of Internet resources, topologies (network throughput and latency), and their load and availability.

Fourth, in distributing the data chunks to a set of workers, the submission node may be a bottleneck as cable/DSL upload speeds are often quite low (for example, often near 133Kbps). We will perhaps need to look into P2P data distribution methods to ensure timely data distribution and completion.

Fourth, one issue in [63] is when to switch to the second phase of RUMR. We will study this issue in detail.

Finally, there may be firewall issues if the submission node is required to contact other nodes in the VM. Methods such as TCP hole-punching [64] may be applied here.

References

[1] Boinc statistics for the world! http://www.boincsynergy.com/stats/index.php. website.

[2] distributed.net. http://www.distributed.net/.

[3] Einsten@home.

[4] Enabling Grigs for E-Science in Europe.

[5] Frontier.

[6] The great internet mersenne prime search. http://www.mersenne.org/.

[7] Hmmer.

[8] Mencoder media player.

[9] Mesh technologies. http://www.meshtechnologies.com/.

[10] Noise ninja.

[11] Ourgrid's website. http://www.ourgrid.org/.

[12] Sensor Networks. `http://www.sensornetworks.net.au/network.html`.

[13] Vfleet volume rendering package.

[14] Xtremweb-ch's website. http://www.xtremwebch.net/.

[15] Yacy - distributed p2p-based web indexing.

[16] The grid, blueprint for a new computing infrastructure. Morgan Kaufmann, 2003.

[17] N. Abdennadher and R. Boesch. A scheduling algorithm for high performance peer-to-peer platform. In *CoreGrid Workshop, Euro-Par 2006*, Dresden, Germany, August 2006.

[18] M. D. A.K. Mok. Multiprocessor scheduling in a hard real-time environment. In *Proc. Seventh Texas Conf. Comput. Syst.*, Texas, U.S.A., 1978.

[19] D. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. In *Europar'02*, pages 197–206, 2002.

[20] D. Anderson. Boinc: a system for public-resource computing and storage. In *Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Pittsburgh, USA, 8 November 2004.

[21] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *proceedings of the 5th IEEE/ACM International GRID Workshop*, Pittsburgh, USA, 2004.

[22] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[23] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2003.

[24] E. B. andSungJin Choi andMaengSoon Baik andChongSun Hwang andChanYeol Park andSoonYoung Jung. Scheduling scheme based on dedication rate in volunteer computing environment. In *Third International Symposium on Parallel and Distributed Computing (ISPDC 2005)*, Lille, France, 2005.

[25] J. Baldassari, D. Finkel, and D. Toth. Slinc: A framework for volunteer computing. In *Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems - PDCS 2006*, Dallas, Texas, USA, November 13-15 2006.

[26] A. Bassi, M. Beck, G. Fagg, T. Moore, J. S. Plank, M. Swany, and R. Wolski. The Internet BackPlane Protocol: A Study in Resource Sharing. In *Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, Germany, 2002.

[27] V. Bharadwaj, D. Ghose, V. Manni, and T. Robertazzi. Scheduling divisible loads in parallel and distributed systems. *IEEE Computer Society Press*, 1996.

[28] J. Blazewicz, M. Drozdowski, and M. Markiewicz. Divisible task scheduling - concept and verification. *Parallel Computing*, 25:97–98, 1999.

[29] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proceedings of SIGMETRICS*, 2000.

[30] S. Bouchenak, N. D. Palma, D. Hagimont, and C. Taton. Autonomic management of clustered applications. In *IEEE International Conference on Cluster Computing*, Barcelona, Spain, 2006.

[31] A. R. Butt, T. A. Johnson, Y. Zheng, and Y. C. Hu. Kosha: A Peer-to-Peer Enhancement for the Network File System. In *Proceeding of International Symposium on SuperComputing SC'04*, 2004.

[32] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky. Computing on large scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Science (FGCS)*, 2004.

[33] H. Casanova. Divisible load scheduling, 2007.

[34] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and Performance of an Enterprise Desktop Grid System. *Journal of Parallel and Distributed Computing*, 63:597–610, 2003.

[35] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the world, unite! ! ! *Journal of Grid Computing*, 2006.

[36] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, 2003.

[37] S. Draves. The electric sheep screen-saver: A case study in aesthetic evolution. In *3rd European Workshop on Evolutionary Music and Art*, march 2005.

[38] M. Drozdowski and P. Wolniewicz. Experiments with scheduling divisible tasks in cluster of workstations. In *Europar'2000*, pages 311–319, 2000.

[39] G. Fedak. *XtremWeb: une plateforme générique pour l'étude expérimentale du Calcul Global et Pair-à-Pair*. PhD thesis, Université Paris Sud, June 2003.

[40] G. Fedak, C. Germain, V. Neri, and F. Cappello. XtremWeb: A Generic Global Computing Platform. In *CCGRID'2001 Special Session Global Computing on Personal Devices*, 2001.

[41] R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis, and W. Freeman. Removing camera shake from a single photograph. *ACM Transactions on Graphics, SIGGRAPH 2006 Conference Proceedings*, 25(3):787–794, 2006.

[42] Y. Fernandess and D. Malkhi. On Collaborative Content Distribution using Multi-Message Gossip. In *Proceeding of IEEE IPDPS*, Rhodes Island, 2006.

[43] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *Proceedings of IEEE/INFOCOM 2005*, Miami, USA, March 2005.

[44] A. Hsu, March 2005.

[45] S. F. Hummel. Factoring: A method for scheduling parallel loops. *Communications of the ACM*, 35(8):90–101, 1992.

[46] A. Iamnitchi, S. Doraimani, and G. Garzoglio. Filecules in High-Energy Physics: Characteristics and Impact on Resource Management. In *proceeding of 15th IEEE International Symposium on High Performance Distributed Computing HPDC 15*, Paris, 2006.

[47] X. P. A. R. C. John F. Shoch, Xerox Palo Alto Research Center; Jon A. Hupp. The "worm" programs - early experience with a distributed computation. *Communications of the ACM*, 3(25), 1982.

[48] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello. Characterizing result errors in internet desktop grids. Technical Report INRIA-HALTech Report 00102840, INRIA, 2006.

[49] D. Kondo, A. Chien, and C. H. Rapid application turnaround on enterprise desktop grids. In *ACM Conference on High Performance Computing and Networking, SC2004*, 2004.

[50] D. Kondo, B. Kindarji, G. Fedak, and F. Cappello. Towards soft real-time applications on enterprise desktop grids. In *Proceedings of 6th Innternational Symposium on Cluster Computing and the Grid CCGRID'06*, Singapore, 2006.

[51] J. Kubiatowicz and all. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[52] O. Lodygensky. *Contribution aux infrastructures de calcul global: délégation inter plates-formes, intégration de services standards et application à la physique des hautes énergies.* PhD thesis, Université Paris Sud, september 2006.

[53] J. Lopez, M. Aeschlimann, P. Dinda, L. Kallivokas, B. Lowekamp, and D. O'Hallaron. Preliminary report on the design of a framework for distributed visualization. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 1833–1839, Las Vegas, NV, 1999.

[54] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT, 2002.

[55] A. Mok. *Fundamental design problems of distributed systems for the hard ral-time environment*. PhD thesis, Massachusetts Institue of Technology, 1983.

[56] A. Mok. The design of real-time programming systems based on process models. In *Proc. IEEE Real-Time Syst. Symp.*, 1984.

[57] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet - the popcorn project. In *International Conference on Distributed Computing Systems 1998*, 1998.

[58] E. Otoo, D. Rotem, and A. Romosan. Optimal File-Bundle Caching Algorithms for Data-Grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 6, Washington, DC, USA, 2004. IEEE Computer Society.

[59] Platform Computing Inc. `http://www.platform.com/`.

[60] PPDG. From fabric to physics. Technical report, The Particle Physics Data Grid, 2006.

[61] J. Pruyne and M. Livny. A worldwide flock of condors : Load sharing among workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[62] D. Qiu and R. Srikant. Modeling and Performance analysis of BitTorrent-like Peer-to-Peer Networks. *SIGCOMM Comput. Commun. Rev.*, 34(4):367–378, 2004.

[63] K. v. d. Raadt, Y. Yang, and H. Casanova. Practical divisible load scheduling on grid platforms with apst-dv. In *IPDPS*, 2005.

[64] A. Rezmerita, T. Morlier, V. Neri, and F. Cappello. Private virtual cluster: Infrastructure and protocol for instant grids. In *Euro-Par*, pages 393–404, 2006.

[65] A. Rodriguez, A. Gonzalez, and M. P. Malumbres. Performance evaluation of parallel mpeg-4 video coding algorithms on clusters of workstations. *International Conference on Parallel Computing in Electrical Engineering (PARELEC'04)*, pages 354–357.

[66] A. Rosenberg. Sharing partitionable workloads in heterogenous nows: Greedier is not better. pages 124–131, International Conference on Cluster Computing (Cluster 2001), 2001.

[67] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 2001.

[68] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima. Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.

[69] L. F. G. Sarmenta and S. Hirano. Bayanihan: Building and studying volunteer computing systems using java. *Future Generation Computer Systems*, 15(5/6), 1999.

[70] M. Shirts and V. Pande. Screen Savers of the World, Unite! *Science*, 290:1903–1904, 2000.

[71] S. Smallen, H. Casanova, and F. Berman. Tunable on-line parallel tomography. In *Proceedings of SuperComputing'01, Denver, Colorado*, 2001.

[72] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[73] United Devices Inc. `http://www.ud.com/`.

[74] S. Vazhkudai, V. F. X. Ma, J. Strickland, N. Tammineedi, and S. Scott. Freeloader:scavenging desktop storage resources for scientific data. In *Proceedings of Supercomputing 2005 (SC'05)*, Seattle, 2005.

[75] B. Wei, G. Fedak, and F. Cappello. Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent. In *The 6th IEEE/ACM International Workshop on Grid Computing, 2005*, Seatle, 2005.

[76] Y. Yang and H. Casanova. Rumr: Robust scheduling for divisible workloads. In *12th IEEE Symposium on High Performance and Distributed Computing (HPDC-12)*, Seattle, June, 2003.

[77] Y. Yang and H. Casanova. Umr: A multi-round algorithm for scheduling divisible workloads. In *IPDPS*, page 24, 2003.

[78] D. Zhou and V. Lo. Wave scheduling: Scheduling for faster turnaround time in peer-to-peer cycle sharing systems. In *Scheduling Strategies for Parallel Processing 2005 (JSSPP'05)*, 2005.