Application Deployment on Catallactic Grid Middleware

Liviu Joita, Omer F. Rana
School of Computer Science and the Welsh eScience Centre, Cardiff University
Queen’s Buildings, 5 The Parade, Roath, Cardiff, CF2 3AA, UK
Phone: +44 (0)29 2087 4812
{l.joita, o.f.rana}@cs.cardiff.ac.uk

Pablo Chacín, Isaac Chao, Felix Freitag, Leandro Navarro
Computer Architecture Department, Technical University of Catalonia
Jordi Girona 1-3, Campus Nord D6
Barcelona 08035, SPAIN
Phone: +34 (93)401-1055
{pchacin,ichao,felix,leandro}@ac.upc.edu

Oscar Ardaiz
Department of Mathematics and Informatics, Public University of Navarra
Campus de Arrosadia, Pamplona 31006, SPAIN
Phone: +34 (948)168076
oscar.ardaiz@unavarra.es

ABSTRACT
An architecture for the integration of Grid applications with a Catallactic middleware is described, along with an analysis of issues associated with this integration in a real world deployment scenario. This approach makes use of the economic concept of “Catallaxy”, which understands a market as a decentralised and self-organizing coordination mechanism, as opposed to a centralised command economy. A prototype is described which consists of an existing Grid application making use of a distributed query service. The Catallactic middleware has been implemented using Globus Toolkit 4, Web Services Resource Framework (WSRF) and WS-Agreement. The evaluation of this prototype shows the feasibility of offering existing Grid applications in a computational market, and demonstrates how the middleware based on market mechanisms can in effect be used to balance query request workload across multiple Grid services.

Category and Subject Descriptors:
C.2.4 [Computer Systems Organization] Distributed Systems

General terms:
Design, Experimentation

Keywords:
Middleware, Grid, Economic-based Allocation

1. INTRODUCTION
There has been significant interest in utilising an economic paradigm for exchanging Grid resources and services [3]. A key motivation behind this approach is the capability to schedule access to services based on a market mechanism, thereby allowing a more fair and efficient approach to sharing resources in high demand. Most existing approaches rely on the existence of a centralised broker that coordinates resource access, which is generally implemented over existing Grid middleware. We propose an alternative approach, based on the Catallaxy mechanism proposed by von Hayek [8], which does not need to support such centralised brokers. Catallaxy makes use of a “free-market” self-organisation approach, which enables prices within the market to be adjusted based on the particular demands being placed on specific scarce services. In previous work [2] we performed simulations of Catallactic Grid markets comparing them to centralized economic allocations; satisfactory evaluation results encourage us to investigate the feasibility of implementing Catallactic Grid middleware and its integration with Grid applications.

The Catallaxy approach is a coordination mechanism based on negotiation and price signalling between decentralized autonomous agents. Catallaxy is a way to inform individual (agent) about the knowledge that may be contained within other agents, and provides an exchange of information that leads to the generation of prices which comply with the value every individual (agent) assigns to this information [2]. (We use the term “agent” to refer to an autonomous service provider or user, having the capability to update and modify the services they offer/use, and to determine how much information about these services should be made accessible to others). Catallaxy therefore leads to the development of self-organizing individuals (agents) that are highly dynamic, thereby leading to systems which behave in a Peer-2-Peer fashion. Such an approach is particularly suited to “Open Systems”, where detailed knowledge about particular agents may not be known apriori.

The Catallactic “free-market” mechanisms can be applied to computational and data Grids [6], in which two interrelated markets appear: the resource and service markets. In the Grid resource market, resource providers sell their computational, storage, bandwidth or tool resources to resource buyers. The traded goods are physical resources which will be used by buyers to execute their own applications. It is expected that in a Grid resource market there are a large number of resource sellers and buyers, thus leading to the requirement of Catallactic “free-market” mechanisms.

In the Grid service market, service providers sell services to service clients. The traded good in a service market are services which provide particular application functionality; such as a transcoding service, a query execution service, a molecule docking service, etc. Each service is therefore required to provide a self-contained functionality, and in the first instance we do not assume any dependencies exist between these external services. A service buyer is interested in using such a service alongside the service(s) that he already has. Assuming a service-rich environment, we expect that a Grid service market has a large number of service sellers and buyers, thus leading to the requirement of Catallactic “free-market” mechanisms.

Service providers can buy resources at the Grid resource market to provide services in the Grid service market. For example, a transcoding service provider can buy computational resources in the Grid resource market to execute its transcoding application for
a particular client request. Although there are some dependencies between the resource and service markets, each can operate somewhat autonomously by applying Catallactic mechanisms.

The objective of this paper is to first describe an architecture for the integration of Grid applications on a market that supports the concept of Catallaxy (generally by integrating the application with a Catallactic middleware). A prototype is described, which is based on a distributed database query application within the Architecture/Engineering/Construction industry [11], [17]. The application makes use of a distributed search strategy to locate building products within distributed product catalogues.

The rest of paper is organized as follows: section 2 presents the architecture of a resource allocation middleware based on the concept of Catallaxy, section 3 introduces a model to integrate Grid applications with the Catallactic middleware. Section 4 presents a prototype combining the COVITE application with the Catallactic middleware, including preliminary evaluation results for the behaviour of the middleware. Finally section 5 contains conclusions and future work.

2. CATALLACTIC MIDDLEWARE

The implementation of Catallaxy in a real world Grid application requires the design of a Catallactic middleware capable of dealing with a variety of different application scenarios. This middleware is intended to support the implementation of Catallactic computational markets offering a set of high level abstractions and mechanisms such as: locating and managing resources, locating other trading agents, engaging agents in negotiations, and adapting to changing conditions. Due to the potential variability in the application characteristics, the middleware should implement only general mechanisms on which application specific strategies and policies can be plugged to adapt to different scenarios. The requirements and technical issues that need to be considered are discussed in this section.

2.1 Application Scenarios

Application scenarios for the Catallactic middleware have the following features:

Dynamic: operate within a changing environment and the need for adaptation to changes is one of the potential areas where Catallaxy can have a competitive advantage.

Diverse: as requests may have different priorities, a response should be returned according to these priorities.

Large: as such applications may involve coordination between a large number of elements, locality is necessary to scale.

Partial knowledge: it is not possible for any entity to have complete information about the state of the system. This can be caused by scale issues, such as a large number of elements or messages, or communication latency (i.e. information arrives too late), which requires locality (thereby leading to scalability problems).

Complex: the applications are characterised by a large number of parameters, and learning mechanisms are necessary to self-adjust or adapt to changes. Therefore, optimal solutions are not easily computable.

Evolutionary: the applications are open to changes which cannot be taken into account in the initial set-up, and able to learn and decide with limited information (about their neighbours, limited historic data, etc.).

2.2 Requirements

The characteristics of the application scenarios discussed above lead to the following requirements for the middleware architecture:

Scalability in highly dynamic environments: the Catallactic middleware should be able to address scenarios with thousands of nodes in a highly dynamic environment, where nodes enter and leave the network frequently. The dynamism in the network configuration implies that information about the system should be maintained at a minimum (avoiding global topological information) and that updates must be easy and efficient.

Handle heterogeneous environments: scale also implies a high level of heterogeneity in applications, the underlying platform, resources, service properties of providers, and availability of nodes (some will be quasi permanent, other will enter and leave).

Compatibility with different base platforms: different base platforms should be supported, thereby leading to the definition of a set of generic APIs. Some adaptors may be needed to translate this generic model to the specific model used by each platform. This translation mechanism could harm the performance of the system if transformations are complex or frequent.

Component self-organization: the dynamicity of the network prevents an apriori configuration of peers, or the maintenance of centralized configuration files. Peers need to continuously discover network characteristics and adapt accordingly, which requires a distribution of important system functions such as security, resource management, topology management, etc – functions which have been traditionally reserved to specialized nodes.

Support different implementation architectures: the middleware may be deployed under different configurations. Each component should therefore not make any assumptions about a specific distribution of functionalities among the physical components of the middleware infrastructure.

2.3 Architecture

A layered architecture provides a clear separation of concern between the layers. This allows construction of a more adaptable system, as the upper layers can be progressively specialized (by means of pluggable rules and strategies) into specific application domains. The following 5 layers are supported:

Application Layer: constitutes end user applications like collaboration tools, problem solving environments, and many others. Applications rely on the base platform for functions like communication and platform level resource management. However, applications can have application level resources, like a virtual meeting room in a collaboration tool or a matrix resolution algorithm in a scientific environment. The interaction model between the application layer and the Catallactic middleware is application and middleware dependent. Applications can interact directly with the Catallactic middleware (becoming Catallactic enabled applications) to manage their resources or they can
interact transparently by means of the base platform they are built on.

**Economic Algorithms Layer:** Implements economics algorithms for resource allocation. These algorithms should be domain and platform independent. This layer includes a set of interacting agent services that play the roles of sellers and buyers in service and resource markets. Also, in this layer are extensions and specializations of the functionalities provided by the underlying framework, to adapt them to the specific ALN and the resource allocation policies in place.

**Economic Framework Layer:** provides primitives that support the implementation of Catallactic algorithms, such as finding peer agents to negotiate, starting negotiation, making a bid, etc. It is dependent on the agent platform being used, but should be independent of the application domain and the base platform. This layer is structured as a set of basic entities that model the interaction of trading agents in a market to exchange goods. These abstract entities are the building blocks of the Catallactic algorithms.

**Peer Agent Layer:** platforms that host the Catallactic agents offer a generic P2P capability for the discovery and communication mechanism, and a programmable interface with the underlying platform. This layer covers the basic functions that will be used by all implementations; it is responsible for interfacing with the underlying platform and extending it where necessary.

**Base Platform Layer:** supports applications and Catallactic middleware, and may be domain specific. The model of interaction with the Catallactic middleware depends on the architecture of the base platform, but in general will require the implementation of a connector, which routes the request for resources to the corresponding economic agents. In some cases, this might require extending core platform components, like the GRAM (Globus Resource Allocation Manager) [7].

The middleware we have designed has the layered architecture shown in Figure 1. The “Applications” layer is for domain specific end user applications like collaboration tools, problem solving environments, etc. Applications interact with the “Grid Market Middleware” in order to obtain the services required to fulfil a particular function. The “Base Platform” supports the applications providing the hosting environment for Grid services.

![Figure 1 - Layered middleware architecture](image)

### 2.4 Middleware Toolkit Selection

The middleware toolkits selection process was carried out taking into account three different but related aspects: potential application scenarios, software architecture, and available middleware toolkits. Considering previously identified middleware requirements, six toolkits were selected and reviewed: DIET and JADE agent platforms, J2SE, WSfR/OGSA, Web Services and JXTA. The evaluation includes their functional properties according to the software architecture defined for CATNETS [4], their technical characteristics and their suitability as a development toolkit. The evaluation [4] concluded that previously explained middleware requirements need a composition of different middleware toolkits. Performance enhancements could be achieved by a light weighted agent implementation as in DIET, interoperability would benefit from a web services based communication and scalability could be achieved by a strong decentralization of key functions, as in JXTA. CATNETS middleware is implemented as a composition of different middleware toolkits (DIET combined with JXTA and WSRF/OGSA), which achieve a good balance between the functional and non functional requirements.

### 3. APPLICATION INTEGRATION

#### 3.1 Application requirements for Catallaxy

In general, different types of applications and users may come together to form a Virtual Organization (VO) in the context of a particular problem. Generally, a VO is formed to combine the expertise of a group of resource providers to solve a single problem, and may be viewed as an abstraction for grouping resource providers in the context of a particular project, and for a particular duration. Grid infrastructure allows the users of a VO to interact until some solution has been obtained for the problem, or the administrator of the VO decides to terminate it. Market-based coordination may be used to identify possible VO participants, based on policy adherence and Service Level Objectives. Such VOs could potentially include resources such as computing systems connected over public communication infrastructure such as the Internet. There could also be possibilities for each participant within a VO to try to maximize their own utility within the market. There, the requirements upon such a coordination mechanism are manifold, i.e. applications and users have different requirements upon auction mechanisms and any associated underlying institutional rules.

A view of actors involved in a generalized application prototype is shown in Figure 2. The User accesses a general application and makes a task execution request. The request is handled by the application, which builds transforms the user request into an application-specific task. To fulfill this request, it is necessary the provisioning of one or more services by the middleware. This resource provisioning, in turn, requires the provisioning of a bundle of resources managed by the Grid platform.

#### 3.2 Application interaction

The interaction between an application and the middleware is described in Figure 3. When a client issues a request, the application determines which services are required to fulfill it. The middleware searches among the available service providers, which have registered their particular service specifications, like contractual conditions, policies and Quality of Service (QoS) levels. When a suitable service provider is found, the application
requirements are negotiated by agents who act on behalf of the service providers as sellers and the application as buyer. Once an agreement is reached between the trading agents, a service instance is created for the application to use.

![Application Use Case](image)

**Figure 2 - Application Use Case**

**3.3 Integrating Grid Application with the Catallactic Middleware**

Figure 4 depicts a detailed view of the architecture, identifying the placement of logical components along the three layers: the application layer, the Catallactic middleware layer and the base platform layer (as discussed in section 2.3). The application must be provided with an interface issue requests for services to the middleware, and use the references to service instances returned by the middleware. Two modules are necessary for a general application to make use of the underlying Grid market middleware:

The **Master Grid Service (MGS)** module: this module provides an interface between the application and an access points to the Catallactic markets. The MGS could be seen as a service interface that is to be implemented by any application that wishes to trade on the market. In concept this module is generic, but an implementation of it needs to be configured with application specific logic, for example the budget used to purchase services, and any particular requirements these services should have. The MGS is connected to the Catallactic Access Point – which are advertised within market registries. An application user therefore needs to find the registry nearest to them (in terms of their IP domain or connection latency) in order to connect to the market.

The **Catallactic Access Point (CAP)** module: the CAP is the second module required by an application to interact with a Catallactic market. The CAP has been implemented as a Web Service that interacts with the application via the MGS. The functionality of the CAP is to provide an entry point for connecting to the Catallactic middleware, to identify the list of complex services available on markets, as well as to exchange service level agreements (template/offer) between the middleware and the application. Two methods are available in the CAP implementation to achieve this objective:

- getAgreementTemplate(): this method provide access to the agreement template hosted by the CAP. The repository can be implemented via a database or file system. The template identifies the parameters that the CAP can understand, thereby requiring the application to make its request using these sets of parameters. In this way, we avoid the need for a CAP to understand any application specific parameters.
- receivedAgreementOffer(): this method is used to receiving the agreement offer, which contains the request parameters needed by application. These request parameters form the basis for discovering a suitable service on the market.

At the middleware layer, a set of agents provide the capabilities to negotiate for services and the resources needed to execute them. The Complex Service Agent acting on behalf of the application, initiates the negotiation. Basic Service and Resource agents manage the negotiation for services and resources, respectively. Also, a Service Factory is provided to create an instance of the service on the execution platform selected during the negotiation process.

Finally, at the Base Platform layer, a Resource is created to manage the allocation of service instances to resources. This resource represents the “state” of the service from the perspective of the middleware (notice, this does not mean that the service is stateful from the perspective of the application.

A detailed description of the sequence of events in the interaction among applications and the Catallactic Grid Middleware is shown in Figure 5. The application requires allocation of services and resources which are not necessarily co-located; therefore a request has to be made to an access point/middleware in which negotiations take place to find and access them.

The flow of information among the logical components can be summarized as follows: a Client issues a request to the application via the Application Service Builder (AppServiceBuilder) module – the invoked application instance that interprets the client request into a service, which in turn requests the execution of the service to the Master Grid Service (MGS). These two modules are part of the “application” box presented in Figure 3. The MGS contacts a Service Access Point (CAP) asking for a WS-Agreement template for such a service. The MGS fills in the template and sends back an Agreement Offer. WS-Agreement concepts are presented in section 4.5.

The Complex Service Agent initiates Catallactic mechanisms to find appropriate Basic Services and Resources. The Complex
Service Agent uses discovery mechanisms implemented in the middleware Peer Agent Layer to locate Basic Service Agents. When a number of Basic Service Agents are discovered, it starts negotiating with one of them. In turn each Basic Service Agent must discover and negotiate with Resource Agents in the resource market to find resources on which the service must be executed. Negotiations are implemented by the Economic Framework Layer, where different protocols can be used depending on the strategy employed by the agent. When an agreement with a Basic Service Agent is reached, the Resource Agent instantiates a Resource to keep track of the allocated resources and returns to the Basic Service Agent a handle for this resource. Consequently
Basic Service Agents use the Service Factory to create an instance of the service on the selected GT4 container. The Basic Service Agent then returns to the Complex Service Agent a reference of this service instance and the related resource(s) that the service makes use of it. The reference to the service is returned to the MSG, which uses it to invoke the service to be executed.

4. PROTOTYPE IMPLEMENTATION

4.1 Cat-COVITE application

To illustrate the use of the Catallactic middleware, we extend an existing application developed in the COllaborative Virtual TEams (COVITE) project [11]. This application is primarily intended for the Architecture/Engineering/Construction industry, where suppliers of building products (such as doors, windows, light fitting, etc) and purchasers (building developers, construction companies) collaborate to procure supplies for a particular construction project. In this application, suppliers make their product catalogues accessible as Web Services, allowing a single purchaser to query these multiple catalogue concurrently. Participants from a number of different organizations need to work collaboratively within this application, as purchasers can act as suppliers to larger construction companies, whereas suppliers need to work with industry associations who develop schemas that suppliers use to advertise their products. These participants work concurrently, thus requiring real time collaboration between geographically remote individuals. Each consortium is in effect a virtual organization (VO). The application permits searching across a number of supplier databases to retrieve products matching a criteria set by the purchasers or contractors. The application enables a search to be conducted, making use of the multiple machines in a network to retrieve the matching products.

The COVITE prototype application is divided into two functional services: Security Service and Multiple Database Search Service (MDSS). The MDSS enables searching across a large number of Supplier Databases (SD), using a single Master Grid Service (MGS) instance, which may in turn interact with a search service hosted on a cluster of machines (with one search service per machine) in a network. In this instance, the query is defined according to a data model that is specific to a given construction project. Arbitrary text queries (as in the Google.com search engine, for instance) are not allowed.

The COVITE application makes use of a centralized MDSS and enables VOs to plan, schedule, coordinate, and share services between designs and from different suppliers. The application prototype has been extended so that it can use a Catallactic market model as a way to discover suitable services – and is referred to as the Catallactic COLlaborative Virtual TEams (Cat-COVITE). It is based on a Service Oriented Architecture (SOA), and consists of three main elements: (i) one or more user services; (ii) a “Master Grid Service” (MGS) - responsible for interacting with the Catallactic middleware to find an end point reference for a service instance, and (iii) one or more service instances that are being hosted on a particular resource. Cat-COVITE applications involve searching through distributed product catalogues - modelled as a Web Services-enabled database, using a distributed search via multiple machines within a cluster. The particular approach adopted in the Cat-COVITE application is employable in other industrial applications which make use of distributed databases. Consequently, the lessons learned from this application, and integration with the Catallactic middleware may find use within a very wide community.

4.2 Catallactic model for Cat-COVITE scenario

Figure 6 shows a Catallactic enabled COVITE (Cat-COVITE) prototype instance and related Catallactic agents as buyers and sellers in the service and resource markets. Cat-COVITE makes use of components of the COVITE application, detailed in [17], which can be mapped to actors in a Catallactic market. The main entity of COVITE prototype is the Master Grid Service (MGS), which uses services from a central broker in order to execute an application task, while in Cat-COVITE, uses the Catallactic Access Point (CAP) entity to find those services on decentralized markets. The entities that comprise the Cat-COVITE instance are: Complex Service, which is an abstract component consisting of the MGS, the CAP – the access point between the application and the market, and the Complex Service Agent; the Query Job Service - a type of Basic Service; and the Resource where the Query Service is executed – as a type of computational resource. We do not consider any service composition mechanisms and concentrate on the allocation process for each Basic Service.

The Complex Service Agent, is the buyer entity in the service market, and the Query Job Service, via the Basic Service Agent (BS Agent in Figure 6), is the seller entity on the service market.

Complex Service Agent starts parallel negotiation with a number of agents representing Query Job Services (Basic Services) and chose one of them based on price negotiation, while the Complex Service, via MGS, translates a request to a Basic Service, of query job service type.

The Query Job Service, as a Basic Service type, involves query execution on a particular database and consists of:

- Query Job Execution Environment (for deployment over machines within a cluster, which are able to execute the query).
- Translation of a query to resource requirements.

Resource seller entities are able to provide a set of resources via the Local Resource Manager (LRM) – accessed via Resource agents.

The Query Job Service is the buyer entity in the resource market, and the LRM are the seller entities on the resource market. The main function of a Basic Service agent within the resource market is co-allocation of resources (resource bundles) by parallel negotiation with different resource providers (LRM entities).

4.3 Physical deployment on GT4 containers

The logical architecture described in the previous section can be implemented in different ways depending on the base platform used. We have made a specific implementation on a GT4 [10] based platform, and based on the following assumptions. We assume that application-specific service have been pre-deployed on a set of GT4 containers. Second, the only “resources” considered in the negotiation are the “rights” to execute the service on a specific container. Finally, the service can be instantiated on a container using a generic factory.

The application and the Master Grid Service (interface with the middleware) are co-located with the Complex Service Agent,
which represents the application in the negotiation process. On each Grid Container (GT4) where the service is deployed, resides the corresponding Basic Service Agent, which negotiates with the Complex Service Agent for access to the service. In this container also resides the Resource Agent, which negotiates with the Basic Service Agent for the rights to execute the service in this container. Finally, a Resource is created as result of the negotiation process, which represents the “rights” to execute the service in this container.

4.4 Middleware implementation

The implementation of the middleware builds on the use of different middleware toolkits, namely the DIET agent platform [5], JXTA peer-to-peer platform [12] and the WSRF/OGSA implementation of Globus Toolkit 4 [10]. DIET provides a modular and lightweight scalable execution platform for agents, JXTA offers a peer-to-peer networking environment and GT4 provides full support for resource management in distributed service-based environments. A detailed description of the selection of middleware toolkit is given in [4].

The middleware is implemented as a set of simple, specialized agents. Framework agents support the basic functions needed to implement economic algorithms, like access to markets, negotiation, good trading, etc. Peer Agent Layer agents implement the low level functionalities to support system execution: overlay network, object discovery, communications. Formation of an overlay network, object discovery and communication is implemented using JXTA protocols to route messages among agent nodes [13]. The management of local resources, in this case services offered by the service providers, is based on the WSRF framework [15] offered by GT4. A detailed description of middleware implementation can be found in [4].

A search request is propagated by the decentralized mechanism implemented using JXTA Peer Resolver Protocol, allowing complex multi-attribute queries using query resolvers registered on each node. Query resolvers make use of the Index Service in GT4 to resolve queries against the specific search attributes and locate the desired services and resources.

4.5 WS-Agreement in Cat-COVITE prototype

WS-Agreement protocol specification has been developed by the GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Scheduling and Resource Management (SRM) Area of the Global Grid Forum (GGF) [9]. WS-Agreement is an XML language protocol for specifying an agreement between a resource/service provider and a consumer [14]. It is generally aimed to be a one-shot interaction, and is not directly intended to support negotiation. However, it can form a useful basis on which negotiation between two parties may be conducted. WS-Agreement forms the basis for choosing between multiple service and resource providers. The service provider acts as the agreement provider, while the service consumer as the agreement initiator.

An agreement consists of several parts, according to the WS-Agreement specification [14]: the agreement name, which is optional; the agreement context includes the parties to an agreement, reference to the service(s) provided in support of the agreement, and the lifetime of the agreement; the agreement terms, which describe the agreement itself, can contain: the service description terms, which provide information needed to instantiate or otherwise identify a service to which this agreement pertains. And the guarantee terms, which specify the service levels that the parties are agreeing to.

Based on the Cat-COVITE application, the WS-Agreement use scenario is as follows: an MGS needs to run a search job, and sends an Agreement Offer (AO), based on the Agreement Template (AT) downloaded from the Catallactic Access Point (CAP), to the CAP to finding a search service. The Complex Service Agent, acting on behalf of the Complex Service (MGS) chosen by the CAP, negotiates with the Basic Service Agents (in the CATNETS environment) for query services to fulfill the job. The agreement template (AT) specifies the service description elements that are allowed by the factory which advertises it. In the
Appendix an agreement template for Cat-COVITE applications which is WS-Agreement compliant is presented. The agreement offer (AO) is initiated by the agreement initiator (in this case the MGS). An agreement offer for Cat-COVITE application is also provided in the Appendix, which is also WS-Agreement compliant. The agreement acceptance is the same as the agreement offer if the agreement provider accepts the conditions of the offer. If the agreement provider does not accept the offer, the agreement initiator has to send an alternative agreement offer (this continues until a consensus is reached, or the initiator or provider terminates the interaction). The negotiation between the agreement provider and initiator, in this scenario, is based on price.

4.6 Negotiation protocol

For this prototype we have used as economic agents the ZIP (Zero Intelligence Plus) agents [13] which use a gradient (ascent/descent) algorithm to set the price for resources. A Complex Service initiates negotiations with a price lower than the available budget, sending a bid to Basic Services. If a Basic Service is willing to accept this bid, it responds by sending a conformity message and allocates the resources it requires to fulfill the request. If a Complex Service is not able to buy at that price, it increases its bids until either it wins or reaches the budget limit.

Basic Service Agents start selling the service at a price which is solely influenced by the resource price as reported by the Resource Agent of the node on which it executes. As Basic Service agents participate in negotiation, the price will also be influenced by demand on the service. If a Service Agent is selling its service, it incrementally increases the price to determine the current market value. When it is no longer able to sell, it lowers the price until it becomes competitive again or it reaches a minimum price defined by the current utilization of the resource.

The price update in the Basic Service follows a gradient algorithm: if the current price is above the maximum offered price of all bids, the Basic service tries to lower its price to become competitive. If the price is below, it evaluates the market by raising the price, looking for higher profits.

The local resource manager in this prototype was intentionally left very limited, as we wanted to evaluate how the economic mechanism alone will impact the resource allocation. The local resource manager only keeps track of the CPU utilization (using information provided by the Linux kernel) and “allocates” it without any warrantee, because services will compete with other workload for the CPU. The Resource Agents calculates the resource price based on the current resource utilization, as reported by the local resource manager, following the pricing model presented in [16].

4.7 Prototype performance assessment

In order to test the performance of the market-based resource allocation mechanism, we setup experiments deploying several instances of the middleware on a farm of Linux servers. Each node has dual CPU Intel PIII, 1 GHz processors with 512 MB of memory. The nodes in are connected by an internal Ethernet network operating at 100Mbs. We deploy the Catalytic middlewate and the application services on three nodes (named arvei-10, arvei-11 and arvei-12). We generate a baseline load on all three nodes of approximately 25% of CPU usage to simulate some background activity.

The experiments consist of launching 3 application clients concurrently from 3 other nodes. Each client performs 50 requests, in intervals of 10 seconds. Whenever a client wins a bid for a service, it invokes the service on the selected node. The complete experiment runs for about 10 minutes. To test the adaptability of the algorithm to changes in the system workload, during the experiment we artificially varied the workload on one of the nodes (arvei-10) up to 95-100% of CPU usage.

Figure 7 shows how the algorithm effectively balances the workload based on price, as utilization level impacts price of the server’s resources – during negotiation a client prefers a service with the lowest price. Figure 8 shows how the negotiation time drops dramatically as agents “learn” how to set prices to become competitive in the market. However, when market conditions change, the negotiation time increases while agents adjust to these new conditions.
5. CONCLUSIONS

The Catallactic middleware, based on the economic concept described by von Hayek, has been presented along with an application that makes use of this middleware. An implementation based on GT4, JXTA and DIET has also been described. The prototype implementation shows how existing applications can be adapted to make use of the Catallactic middleware, thereby providing a resource allocation mechanism capable of load balancing and adaptation to changing environments.

We observed that the negotiation protocol will need to go beyond the current WS-Agreement specification to handle the complexities of the bargaining process. Also, WSRF specifications are still too general and do not offer a clear approach for managing virtual resources. The use of standard specifications, such as WSRF and WS-Agreement, provide the main incentives for other application users to make use of our Catallactic market implementation. By providing interaction with the market via an access point (the CAP), we ensure that the application does not need to be modified internally. In addition, the peer-to-peer nature of the Catallactic market model (as opposed to other market models proposed in Grid computing that make use of centralized brokers) makes our approach more suitable to distributed decentralized systems, such as computational and data Grids.

6. ACKNOWLEDGMENTS

This work was supported in part by the European Union under Contract CATNETS EU IST-FP6-003769. This is an extended version of the paper that appears at MGC2005.

7. REFERENCES


APPENDIX: Agreement Template and Offer

<?xml version="1.0" encoding="UTF-8"?>
  <wsag:TemplateName>QueryComplexService</wsag:TemplateName>
</wsag:AgreementTemplate>
</wsag:Context>
<wsag:Terms>
  <BasicServiceName>QueryBasicService</BasicServiceName>
  <NumberOfBasicServiceNodes>0</NumberOfBasicServiceNodes>
</wsag:Terms>
  <wsag:Name>QueryComplexService</wsag:Name>
  <wsag:Context>
    <!-- can be a URI or a security identity of the initiator -->
    NameOfTheInitiator
  </wsag:Context>
  <wsag:ExpirationTime>DateTime</wsag:ExpirationTime>
  <wsag:Terms>
    <BasicServiceName>QuerygBasicService</BasicServiceName>
    <NumberOfBasicServiceNodes>1</NumberOfBasicServiceNodes>
    <BasicServiceConstraints>
      <BasicServiceType>Architectural/Engineering/Construction</BasicServiceType>
    </BasicServiceConstraints>
    <Price>100</Price>
  </wsag:Terms>
</wsag:AgreementOffer>