Lehrstuhl für
Wirtschaftsinformatik

Information Systems
Management

No. 11

2006

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik

Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freitag, Manuel Medina, Leandro Navarro, Miguel Valero (Universidad Polytecnica de Catalunya), Omer F. Rana, Liviu Joita (Cardiff University), Torsten Eymann (University of Bayreuth)

Proof-of-Concept Application - Annual Report Year 2

Bayreuth Reports on Information Systems Management

UNIVERSITÄT BAYREUTH

ISSN 1864-9300

Die Arbeitspapiere des Lehrstuhls für Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The Bayreuth Reports on Information Systems Management comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Alle Rechte vorbehalten. Insbesondere die der Übersetzung, des Nachdruckes, des Vortrags, der Entnahme von Abbildungen und Tabellen – auch bei nur auszugsweiser Verwertung.

All rights reserved. No part of this report may be reproduced by any means, or translated.

Authors:

Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freitag, Manuel Medina, Leandro Navarro, Miguel Valero (Universidad Polytecnica de Catalunya), Omer F. Rana, Liviu Joita (Cardiff University), Torsten Eymann (University of Bayreuth)

Information Systems and Management Working Paper Series

Edited by:

Prof. Dr. Torsten Eymann

Managing Assistant and Contact:

Raimund Matros
Universität Bayreuth
Lehrstuhl für Wirtschaftsinformatik (BWL VII)
Prof. Dr. Torsten Eymann
Universitätsstrasse 30
95447 Bayreuth
Germany

Email: raimund.matros@uni-bayreuth.de

ISSN 1864-9300

IST-FP6-003769 CATNETS

D3.2

Annual Report on WP3: Proof-of-Concept Application (T0+24)

Contractual Date of Delivery to the CEC:	**31. August 2006**
Actual Date of Delivery to the CEC:	**13. October 2006**
Author(s):	Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freitag, Liviu Joita, Manuel Medina, Leandro Navarro, Omer F. Rana, Miguel Valero
Work package:	WP3
Est. person months:	
Security:	public
Nature:	final
Version:	1.0
Total number of pages:	98

Abstract:
This deliverable describes the work undertaken in Work Package 3 of the CATNETs project, which involves the following tasks: Task 3.1: "Implementation of additional services for economic enhanced platforms in Grid/P2P platform", Task 3.2: "Framework architecture/implementation for economic enhanced platforms", Task 3.3: "Performance measuring components for experiments", and Task 3.4 "Distributed application to execute an economic-enhanced Grid/P2P platform and middleware integration"
The document is divided in ten parts: the introduction into the Application Layer Network; Catallactic resource allocation model and middleware architecture; use cases for services models; application scenarios, as generalisation, as well as detailed in two cases: Query service and Data Mining services; middleware implementation; prototype evaluation; future work; relation to other work packages; related work and efforts; and the conclusion and further work.

Keyword list: (optional)

CATNETS Consortium

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-FP6-003769. The partners in this project are: LS Wirtschaftsinformatik (BWL VII) / University of Bayreuth (coordinator, Germany), Arquitectura de Computadors / Universitat Politecnica de Catalunya (Spain), Information Management and Systems / University of Karlsruhe (TH) (Germany), Dipartimento di Economia / Università delle merche Ancona (Italy), School of Computer Science and the Welsh eScience Centre / University of Cardiff (United Kingdom), Automated Reasoning Systems Division / ITC-irst Trento (Italy)

University of Bayreuth
LS Wirtschaftsinformatik (BWLVII)
95440 Bayreuth
Germany
Tel: +49 921 55-2807, Fax: +49 921 55-2816
Contact person: Torsten Eymann
E-mail: catnets@uni-bayreuth.de

Universitat Politecnica de Catalunya
Arquitectura de Computadors
Jordi Girona, 1-3
08034 Barcelona
Spain
Tel: +34 93 4016882, Fax: +34 93 4017055
Contact person: Felix Freitag
E-mail: felix@ac.upc.es

University of Karlsruhe
Institute for Information Management and Systems
Englerstr. 14
76131 Karlsruhe
Germany
Tel: +49 721 608 8370, Fax: +49 721 608 8399
Contact person: Daniel Veit
E-mail: veit@iw.uka.de

Università delle merche Ancona
Dipartimento di Economia
Piazzale Martelli 8
60121 Ancona
Italy
Tel: 39-071- 220.7088 , Fax: +39-071-220.7102
Contact person: Mauro Gallegati
E-mail: gallegati@dea.unian.it

University of Cardiff
School of Computer Science and the Welsh eScience Centre
Cardiff University, Wales
Cardiff CF24 3AA, UK
United Kingdom
Tel: +44 (0)2920 875542, Fax: +44 (0)2920 874598
Contact person: Omer F. Rana
E-mail: o.f.rana@cs.cardiff.ac.uk

ITC-irst Trento
Automated Reasoning Systems Division
Via Sommarive, 18
38050 Povo – Trento
Italy
Tel: +39 0461 314 314, Fax: +39 0461 302 040
Contact person: Floriano Zini
E-mail: zini@itc.it

Changes

Version	*Date*	*Author*	*Changes*

1 Introduction **6**

1.1 Application Layer Network *6*

2 Catallactic Resource Allocation **8**

2.1 Catallactic Resource Allocation Model *8*

2.2 Design Requirements for middleware *9*

2.3 Middleware Architecture *10*

2.4 Integration of Grid Applications *10*

3 Use Cases **12**

3.1 Single Service *12*

3.2 Multiple Services *13*

3.3 Workflow and Bundles *14*

3.4 Generalising *15*

4 Application Scenarios **17**

4.1 Query Services *17*
4.1.1 Cat-COVITE prototype 17
4.1.2 Cat-COVITE prototype and the Catallactic Grid Markets 18
4.1.3 WS-Agreement in Cat-COVITE Prototype 21

4.2 Data Mining Services *22*
4.2.1 Cat-Data Mining prototype 23
4.2.2 Data Mining Services and Grid Markets Middleware (GMM) 24
4.2.3 WS-Agreement in Cat-DataMining Prototype 27

4.3 Generalising Application Scenarios *29*
4.3.1 Application requirements for Catallaxy - generalization 29

5 Middleware Implementation **33**

5.1 Implementation toolkits *33*
5.1.1 Middleware implementation toolkits 33
5.1.2 Development toolkits 34
5.1.3 Testing toolkits 35

5.2 Detail component implementation *38*
5.2.1 Middleware implementation 38
5.2.2 Middleware API 38
5.2.3 Resource Management integration 41

5.3 Deployment and test framework *41*
5.3.1 Deployment framework 41
5.3.2 Test framework 44

6 Prototype evaluation **47**

6.1 Introduction *47*
6.1.1 Objectives 47
6.1.2 Economic Algorithms 47
6.1.3 Data Mining Services scenario 48

6.2 Experimental setup ... *48*
6.3 Performance Results ... *50*
6.4 Evaluation of an uncontrolled environment ... *52*

7 Future Work ... **54**

7.1 Triana Integration ... *54*
7.1.1 Deployment Scenarios ... 54
7.2 Alternative P2P overlays for Middleware ... *62*

8 Relation to other WPs ... **64**

8.1 WP1 ... *64*
8.2 WP2 ... *64*
8.3 WP4 ... *65*

9 Related Work and Efforts ... **66**

9.1 GGF, W3C and OASIS ... *66*
9.2 GRAAP Working Group ... *66*

10 Conclusion ... **68**

10.1 Catallactic-based Grid Middleware ... *68*
10.2 Applications using Catallactic-based middleware ... *68*
10.3 Open issues and future work ... *69*

References ... **70**

Annex A ... **72**

A.1 GMM scripts ... *72*

Annex B – CATNETS Application Repositories Settings ... **78**

B.1 Cat-COVITE prototype ... *78*
B.2 Cat-DataMining prototype ... *83*
B.3 Metrics for the Prototype ... *84*

Annex C – Triana Integration ... **86**

C.1 Cat-COVITE prototype and Triana ... *86*
C.2 Cat-DataMining prototype and Triana ... *90*
C.2.1 Reference to the use case described in section 3.1 ... 90
C.2.2 Reference to the use case described in section 3.2 – Case 1 ... 94

1 Introduction

This deliverable describes the work undertaken in Work Package 3 of the CATNETs project, which involves the following tasks:

- Task 3.1: “Implementation of additional services for economic enhanced platforms in Grid/P2P platform”,
- Task 3.2: “Framework architecture/implementation for economic enhanced platforms”,
- Task 3.3: “Performance measuring components for experiments”, and
- Task 3.4 “Distributed application to execute an economic-enhanced Grid/P2P platform and middleware integration”

The proof-of-concept application used to demonstrate the ideas being developed in these tasks are also outlined in this deliverable.

The document is divided in ten parts: the introduction into the Application Layer Network; Catallactic resource allocation model and middleware architecture; use cases for services models; application scenarios, as generalisation, as well as detailed in two cases: Query service and Data Mining services; middleware implementation; prototype evaluation; future work; relation to other work packages; related work and efforts; and the conclusion and further work.

1.1 Application Layer Network

An Application Layer Network (ALN) is an abstraction that integrates different overlay network approaches, like Grid and Peer-2-Peer systems, on top of the physical connectivity provided by the Internet. A common characteristic of ALNs is the redundant, distributed provisioning and access to data, computation or application services, while hiding the heterogeneity of the service and resource network from the user. One of the main issues in the development of future ALNs is the efficient exploitation of services and resources available in the network.

A key requirement of ALNs is to support scalable, dynamic, and adaptable allocation mechanisms. This issue is being addressed by a number of (on going) Grid and P2P projects such as Globus [FOSTER03], Legion [CHAPIN99], Nimrod/G [BUYYA00], CATNETS [EYMANN05], and Gnutella [ADAR00]. Although considerable progress has been made developing software architectures and allocation methods, which allow clients to obtain services “on demand”, the evaluation of these methods with respect to each other is rarely undertaken. What is missing is a standard set of metrics that could be used as the basis for such an evaluation. It is important that such metrics allow comparison of both the application “behaviour”, along with the infrastructure behaviour.

Currently, no metric framework exists that takes into account the characteristics of applications that may be deployed over ALNs – especially ones that measure the performance of the resource allocation strategy being used. It is assumed that resource allocation is being undertaken in a “market” based environment – allowing service users and providers to sell and buy services and resources. It is also assumed that two concurrent markets coexist – a

"resource market" and a "service market". Hence, once a given service instance has been identified, it may be provisioned on a number of possible resources (each instance of the service would therefore have a different overall cost). The presence of a market allows us to make use of economic concepts with reference to the allocation strategy being adopted. Current Grid and Peer-2-Peer applications often use a service-oriented architecture, which is characterized by dynamic and heterogeneous resources. In such environments one of the key issues is the assignment of resources to services. The characteristics of emerging applications in ALNs define particular resource allocation requirements which have the following characteristics [MONTRESOR03]:

- Dynamicity: There are changes within the environment in which services exist, and there is a need for adaptation to these changes.

- Diversity: Requests may have different priorities, and responses should be assigned according to these priorities. It is therefore possible for a number of different types of requests to co-exist, and determining how these should be handled becomes significant.

- Large scale: There is often a large number of nodes, and it is necessary to group these nodes to create hierarchical organizational structures.

- Partial knowledge: It is not possible to determine at any point in time (or apriori) the full state of the system.

- Complexity: Learning mechanisms are necessary to adapt to changes in infrastructure, and optimal solutions are not easily computable.

- Evolutionary: The system is open to changes which cannot be taken into account in the initial setup. The goal of the evaluation framework presented here is to define a general set of metrics for performance analysis, which can be used in projects evaluating and comparing the performance of the resource allocation mechanisms with the characteristics above. Such a framework should include both technical parameters (such as quality of service factors – like response time) and economic parameters (such as overall cost of computation or data access). This is required to compare different resource allocation methods with each other.

2 Catallactic Resource Allocation

2.1 Catallactic Resource Allocation Model

Current Grid Computing architectures exhibit fairly static resource infrastructure which is connected by physically stable links. The shift to a pervasive Grid, that could exist ubiquitously, demands for a more dynamic consideration of resources and connections. Figure xxx shows our perspective of a Catallaxy-based Grid Market. The market is understood to be a decentralized control mechanism for services and resources. Figure 2-1shows this service oriented model.

A complex service could be represented by a proxy which needs (remote) basic service capabilities for execution – with support for a service selector instance. Complex services are therefore shielded from the details of the resource layer. A basic service is split into the basic service logic and a resource allocator. The logic is able to negotiate with the complex service and to translate the requirements for service execution on a resource instance (e.g. CPU, and storage, etc.). A resource allocator gets the resource specification and broadcasts the respective demand to the local resource managers. This comprises bundles and coallocative negotiations. Bundles are understood as an n-tuple of resource types (e.g. a 3-tuple would be: CPU, storage, and bandwidth); co-allocation describes obtaining resources for one single service transaction from various local resource managers simultaneously. Local resource/job scheduling is not the focus of the project. It is expected that a local resource manager hides all details of the allocation.

On the first market, complex service and basic service negotiate; an agent managing a complex service acts as a buyer, the basic service agent as a seller. The same market roles can be found at the resource layer, the resource allocator is the buyer agent, the local resource manager acts as a seller agent. Contemplating the second market, it is a “n” to “k” market: “n” basic service copies can bargain with “k” resource services. This takes dynamic resources into account. Resources are in our view entities that can fail like basic services, and which are subject to maintenance and inspection procedures or link failures.

The Basic Service is a standardized service for query job execution. At the application layer, the Basic Service consists of:

- a seller entity on the service market;
- an entity to translate a query based on resource demands.

The main functionalities of the Resource Co-Allocator at the resource layer are:

- Representing the buyer entity on the resource market;
- Co-allocating resources (resource bundles) by parallel negotiation with different resource providers (local resource manager entities);
- Informing the Basic Service Logic about the outcome of the resource negotiation.

Entities of a seller on the resource market are able to provide a set of resources via the Local Resource Manager (LRM). The Resource Agents act on behalf of these LRMs, which hide the physical resources behind them.

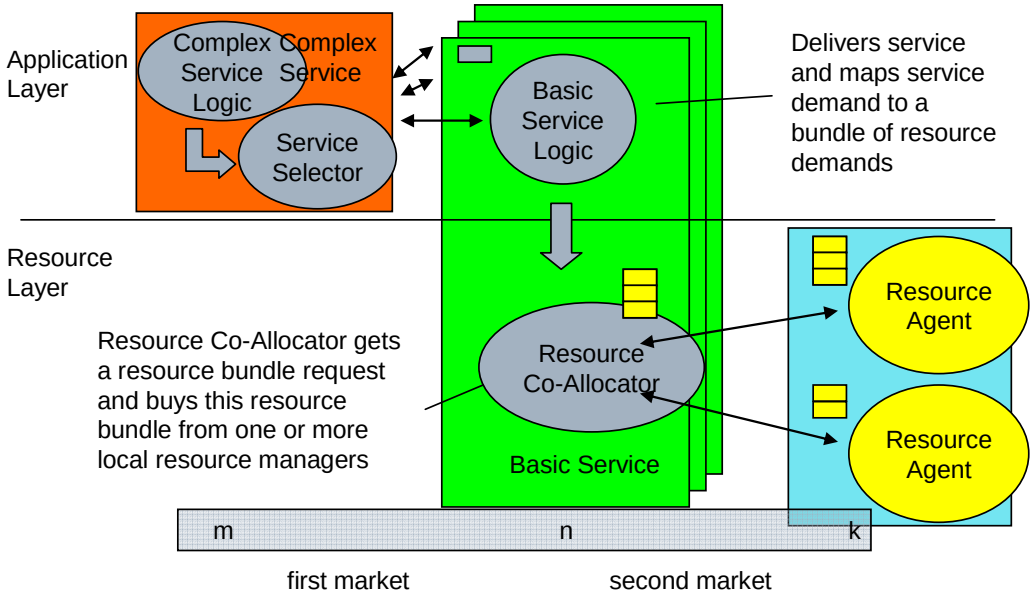

Figure 2-1 Catallactic Service Oriented Model

2.2 Design Requirements for middleware

We expect an ALN to be built from basic services that can be dynamically combined to form value-added complex services. These basic services require a set of resources, which need to be co-allocated to provide the necessary computing power.

Therefore, the introduction of new services into an ALN, due to the dynamic nature of the environment, precludes any manual or static configuration and demands a self-organization approach, where services should be able to self-configure, self-optimize and self-heal [WHW+04].

One goal of self-managed network services is to move away from individual system configuration management to policy management. This approach brings a higher level of abstraction to management, by introducing a policy from which the configuration is derived, allowing components of the infrastructure to apply these derived configurations to the individual systems across the environment.

In this context, the resources associated with self-managing services make use of Service Level Agreement-based (SLA) policies for services and resources, mapping required SLAa to resource needs, discovering resources that guarantee an adequate Quality of Service (QoS), allocating resources to ensure that allocation policies are meet, and providing a management interface to monitor a control service life-cycle. Because of the dynamicity of the environment for which our approach is intended, the service allocation framework must address some specific issues:

- Situatedness: a service must be aware of its location and the availability of peer services within its “vicinity”, with which to collaborate;

- Dynamic (re)configuration: usage patters from service users are unpredictable, therefore neither the location nor the number of service instances could be known in advance. New instances must be created and located as needed;

- Topology neutrality: services deployed in the ALN could have very different interaction topologies. Some will be structured in a hierarchical overlay, like content distribution networks, while others interact in a closely connected P2P overlay;

- Autonomy: a service and the resources it uses will span multiple administrative domains, so each resource provider should be allowed to take decisions autonomously.

2.3 Middleware Architecture

The economic-based resource allocation model has been implemented in the Grid Market Middleware (GMM), which provides the mechanisms to register, manage, locate and negotiate for services and resources. It allows trading agents to interact with each other based on their requirements/demands and engage in negotiations. Furthermore, the middleware offers a set of generic negotiation mechanisms, on which specialized strategies and policies can be dynamically plugged in. The middleware – as shown in Figure 2-2 – has a layered architecture, which allows a clear separation of platform specific concerns from the economic mechanisms, to cope with highly heterogeneous environments. A detailed description of the middleware architecture can be found in [Ardaiz05].

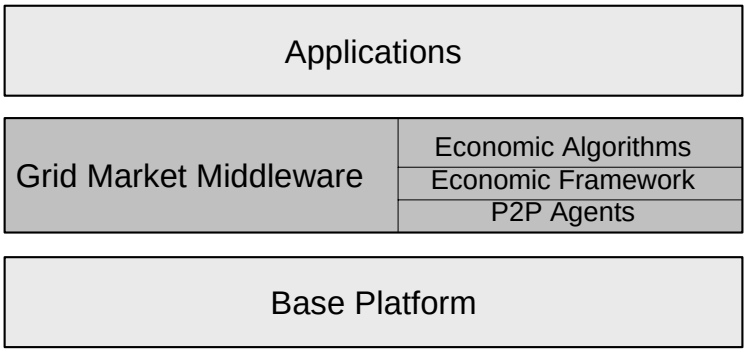

Figure 2-2 Layered middleware architecture

2.4 Integration of Grid Applications

Figure 2-3 shows the interaction between the applications and the GMM, which is based on the WS-Agreement specification [WSAG05]. The use is initiated from within the prototype, at the application level; the interaction between the application and the middleware is as follows: a user issues an application request, which is interpreted by the prototype and converted into an application service request; then the application determines which Grid service(s) is (are) required to fulfil the specific application service request. These Grid services represent either software services (e.g. query services or data mining services) or computational resources. The application service translates these requirements into a WS-Agreement template, which is submitted to the GMM.

The GMM searches among the available Grid/Web Services, which have registered their particular service specifications, such as: contractual conditions, policies and QoS levels. When a suitable Grid service provider is found, the application requirements are negotiated within the middleware by agents who act on behalf of the service providers as sellers and the

application as buyers. Once an agreement has been reached between the trading agents, a Grid service instance is created and a reference is returned to the application, which then can invoke it.

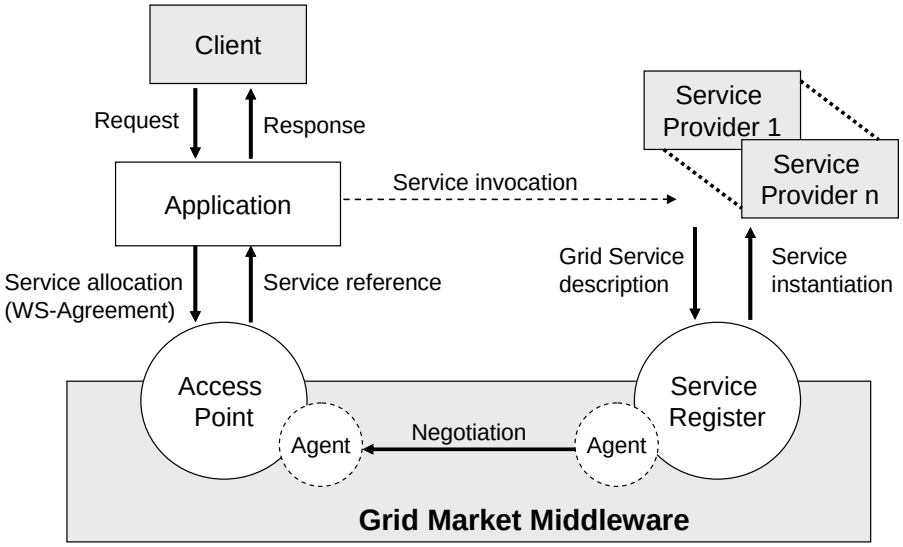

Figure 2-3 Interaction between the Application and the Grid Market Middleware

To take full advantage of these market mechanisms, it will be necessary to include some logic at the application level by specifying a maximum price for a "Web Service type" the application is interested in. This maximum price is like a budget for the buyer service agent that could be spent to buy basic services and resources. The logic at the application level will include a policy document on which the decision is taken by the application to accept or reject the Web Service instance that has been found on the market. The policy document in this case will take the "price" as the characteristic to be taken into consideration, and this is achieved by checking that this is less than the budget proposed initially by the application.

3 Use Cases

The interaction between the Application and the Catallactic GMM can be divided into the following use case scenarios:

1. In the first scenario, the application request is of one complex service type made up of just one basic service type -- Grid/Java/.NET service, without any specifications with regards to characteristics of the service or budget. This scenario will allow to assess the basic functionalities of the Catallactic GMM, such as markets functionalities and agents' negotiations.

2. A more realistic second scenario requires including some logic at the Application level by specifying a maximum price for a complex service type made up of just one basic service type -- Grid/Java/.NET service type, the application is interested in. This maximum price is a budget proposed by the buyer, in this case the complex service agent. This budget represents the price up to which basic services and resources should be purchased. The logic at the application level could be included into a policy document on which the decision is taken by the application to accept or reject the Grid/Java/.NET service instance that has been found on the market. The policy document in this case will take the "price" as the characteristic to be taken into consideration, and this is done by checking that this is less or equal to the budget proposed initially by the application.

3. A more realistic, but complex, scenario includes more characteristics of the complex service type to be specified at the application level, such as bundles of type {price, (CPU, memory, latency)}, while the Middleware response is as a bundle of {price, (CPU, memory, latency), ServiceReference}, where ServiceReference can be: the End Point Reference(EPR) – the address/URI of the Grid/Java/.NET service instance or the location of the WSDL document associated with the service (Java/Grid/.NET). A more complex policy document at the application level can be invoked, as a decision has to be taken by the application to accept or reject the offer from the Catallactic GMM. This complex policy document has to take into account all parameters of the bundle and decide to accept or reject the offer from the market.

An open issue involves investigating the service workflow scenarios that could potentially be required by an application. There are multiple scenarios which involve the use of workflow techniques for service composition, where the use of the market mechanism could be important. The logic of the service workflow resides at the application level, as the application knows what services are needed in order to fulfill an application task. A variety of possible cases to illustrate some of the service composition – de-composition issues are likely to be considered in the future.

3.1 Single Service

In the first case -- a user wants a complex service of type "S" made up of just one basic service "S1", in which case the application requests the Catallactic GMM to find this specific complex service. This case is the simplest and does not require co-allocation of services. The application gets the service instance back from the GMM or nothing, in which case the interaction between the application and the GMM is terminated, and a new request has to be issued by the application.

In a workflow scenario, a client wants a complex service of type "S" (made up of just one basic service "S1"). The MGS asks CAP what complex services of of type "S" are currently available. The CAP subsequently sends a list of possible types of complex services to the MGS -- one of these is "S1". If there is no "S1", the process terminates, requiring the client to make a new request.

In another version of this scenario, an MGS asks for a service of type "S". By using the middleware complex service agent, a basic service that implements "S" is found (referred to as "S1"). If "S1" is available on multiple resources (R), one R that can host "S1" is returned as an end point reference – using the middleware. It may be assumed that different instances of "S1" on different resources have a different price.

Two proof-of-concept prototypes, based on the above scenarios, have been implemented as part of this WP:

- In CAT-COVITE, there is only one S (Query Complex Service), made up of a query basic service (S1). There is only one type of resource (a database hosted on a particular machine). Hence the choice is based on selecting one instance of S1 and one instance of R (all of which do the same thing).

- In Cat-DataMining, there is a complex service S (Data Mining Complex Service), made up of a Basic Service S1 (J48 data mining classifier basic service). The type of resource R is defined by the bundle of {CPU, memory, latency}. The choice is based on selecting one instance of S1 and one instance of bundle R.

3.2 Multiple Services

In the second case, a user wants a complex service S made up of multiple basic services (S1, S2, …, Sn), with or without a specific order.

Case 1. In the case of requirement of a complex service S made up of multiple basic services, without a specific order, the application will request each basic service individually to the GMM. The co-allocation issue then arises at the Application level, where the services will be allocated and a service process workflow is constructed by the user.

Here a client wants a complex service S made up of a set of basic services of types S1, S2, ..., Sn. The client knows the order in which these services should be executed -- but not the MGS and CAP.

The MGS asks the CAP for each Si ($1<i<n$) separately (leading to a Single Service -- see section 3.1). Each service is then requested in the same way as described above.

A proof-of-concept prototype is currently being considered for this scenario for year 3 of the project. In the Data Mining prototype, there are multiple data mining services, such as: J48 classifier (based on the C4.5 decision tree construction algorithm), a Simple K-Means Clustering algorithm, and an Entropy Maximization (EM) Clustering algorithm. There are also multiple converter services that convert the users' input data format to a specific data format that data mining service require. The converter services are the following: Comma-Separated Values to Attribute Relationship File Format (CSV2Arff), C45toArff, Arff2CSV, Arff2C45, where CSV, C4.5 and ARFF are files format that are used by the WEKA data mining libraries [WEKA]. An example of a workflow process would therefore be the

following: a clients CSV data format → Converter CSV2Arff service→ J48 data mining service → visualization service.

Case 2. In this case, a user requests a complex service S made up of multiple basic service S1, S2, …, Sn, with a specific order. It is now necessary for the GMM to carry out service discovery in such a way that either all basic services are found, or nothing is reported to be found. At the GMM level, the negotiations have to be able to coordinate multiple service requirements.

Example: A user needs a Math complex service made up of 4 basic services: addition, subtraction, multiplication and division. At the GMM level, the Math complex service agent has to negotiate with different agents of these four basic service types and try to find at least one basic service of each type in order to fulfill the requirement of a Math complex service.

The workflow process is as follows: a client wants a complex service of type S made up of basic service types S1, S2, ..., Sn. In this case the Complex Service agent does not map to a single service, as in case 1 (see section 3.1), but must be able to coordinate multiple negotiations with different basic services agents in order to fulfil the requirements of the workflow process.

For example, consider the workflow: S1 → S2 → S3, this implies that a complex service must find basic services that implement either all or a subset of this workflow. If there is a basic service for each Si, then the complex service needs to do a co-allocation across 3 basic services. The following may be assumed:

- There is a single basic service per resource, therefore the same argument of resources as in Case 1 applies.

- Multiple basic services per resource, therefore a job scheduler needs to coordinate the execution of multiple basic services on the same resource.

In the Data Mining prototype example for case 2, the workflow process example Converter CSV2Arff Service → J48 Classifier Data Mining Service has to be mapped to a complex service. To be able to generate results from such a complex service, it is necessary to coordinate between the two basic services (the Converter and the Data Mining service), requiring co-allocation between these basic services. This use case will be considered in year 3 of the project.

3.3 Workflow and Bundles

This section describes enhancements to the use cases presented in section 3.2, but with resource bundles -- in this case, each basic service does not run only on one resource, but needs a bundle of resources to execute successfully. Hence, a basic service needs to achieve co-allocation on a number of resources. This is a similar problem to the use cases presented in section 3.2 above, but without any reference to ordering.

A basic service needs a bundle of (R1, R2, ..., Rm) resources and needs all of these resources to be available. At present, we consider a limiting case where m = 3, i.e. a bundle consisting of {CPU, memory and network latency} is used. Currently, the basic service is mapped to a resource R (a single resource), which is expected to provide all of CPU, memory and network latency.

We address two different resource usage scenarios:

- In the first scenario, the interaction between an Application and the GMM is achieved via the WS-Agreement document, and which will only contain the requested service type – equivalent to a complex service type in the CATNETS scenario.

- The second scenario involves the use of a WS-Agreement document, the complex service type and a maximum budget the application is willing to pay to the complex service for allocation. This scenario, enhanced with the possibility of specifying more attributes by an application user, will be further extended in year 3 of the project.

3.4 Generalising

There is an increased interest in distributed applications where services and resources are to be bought from markets or offered by different suppliers.

An example of such a scenario in a real business context is presented by Sun Microsystems. The company presented in 2005 its vision for network computing and unveiled the elements of its new SUN Grid utility offering. Sun tries to simplify computing by enabling customers to tap the power of the network -- using only what they need, when they need it -- for $1/cpu-hr. This offer is still pilot in US. As the IT industry continues to evolve, customers might move away from building data centers in a one-off customized model, toward a standardized model and eventually to a utility model. The use of a standards-based approach will enable them to utilize offerings from different technology vendors, and at the same time, allow ease of expansion if the system needs to grow in the future. Sun's $1/cpu-hr compute and storage offerings provide an interesting use of sharing compute and data resources [SUN].

Sun's expanded vision and new technology innovations represent a significant evolution in network computing, and serves as a pointer for new service-oriented data centers -- spanning the availability of true “information technology on tap” from the network to agile data centers.

A generalised case represented via a UML sequence diagram is shown in Figure 3-1. The application requires allocation of services and resources which are not necessarily located on-site; therefore a request has to be made to an access point/middleware in which negotiations take place for finding and allocating resources. Additional details are provided in section 4, in which two application scenarios are detailed, as well as a discussion of the generalisation of these applications.

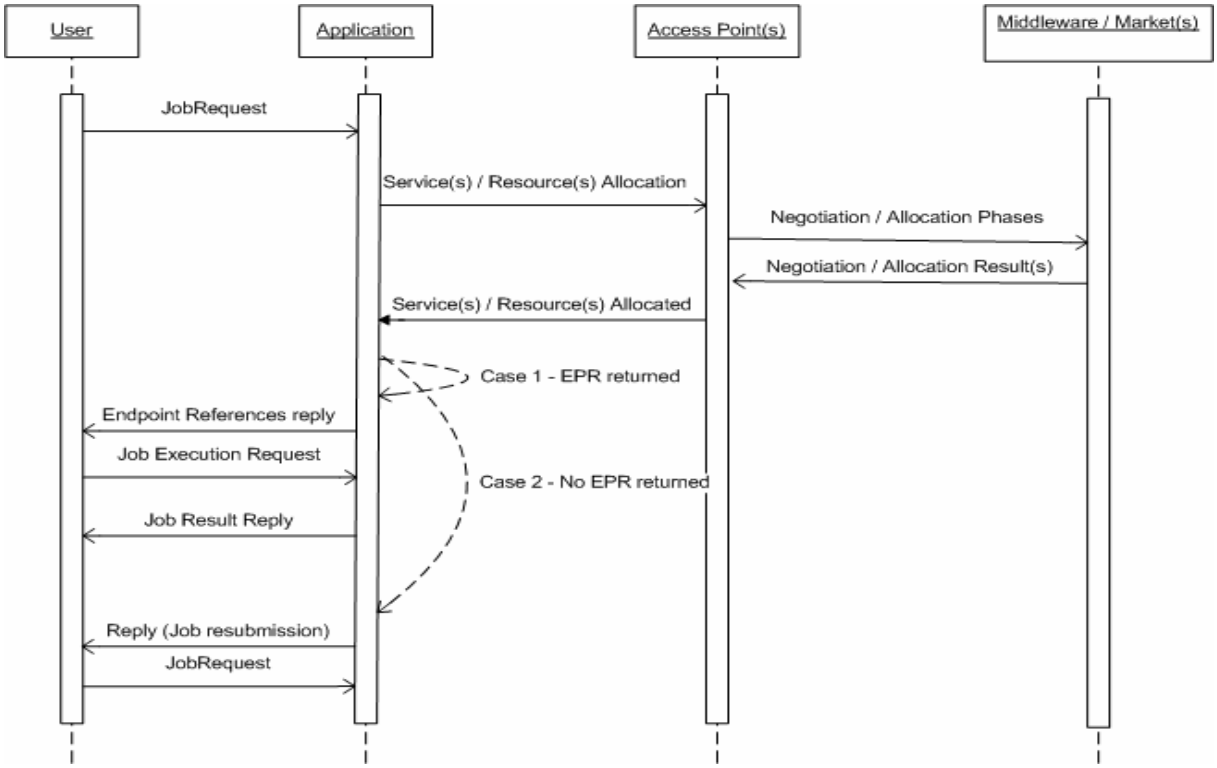

Figure 3-1– Generalising use case scenarios - UML Diagram

4 Application Scenarios

Different examples of application scenarios can be constructed and could benefit from using the GMM in combination with auctions over Grid computing infrastructure. This leads to an advantageous flexibility by allowing different application-specific requirements and needs of services and resources to be addressed. Let us consider one application scenario that requires a highly specialized service (e.g., a medical simulation service, visualization service or query service). Another application requires a specific data mining service or mathematical service. The data mining service is more or less standardized and there are several suppliers offering this service. Different types of auction algorithms could be used by the supplier, such as MACE [MACE06] or a double auction. The medical simulation service, however, does not have many suppliers. As such, the liquidity of the market trading such services may be low. In such cases, English auctions may be useful.

The following sub-sections contain a description of two proof-of-concept prototypes in which Query services and Data Mining services are traded on the service market. Based on experience gained from these prototype applications, we also provide a generalisation of the concepts involved, identifying how other applications could benefit from the Catallaxy mechanism.

4.1 Query Services

An application example which could benefit from an auction-based economic model is an extended version of the COllaborative VIrtual Teams (COVITE) prototype [COVITE04] primarily intended for the Architecture/Engineering/Construction industry – referred to as the Catallactic COllaborative VIrtual TEams (Cat-COVITE) [CatCOVITE05]. This prototype application 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 a GMM to find an end point reference of a service instance, and (iii) one or more service instances that are being hosted on a particular resource. Cat-COVITE application involves searching through distributed product catalogues - modelled as a Web Services-enabled database, and using a distributed search strategy. The particular approach adopted in the Cat-COVITE application is employable in a significant number of other industrial applications which make use of distributed databases. In this way, the lessons learned from this application, and integration with the GMM may find use within a very wide community.

4.1.1 Cat-COVITE prototype

The interaction between the application prototype and the GMM is based on WS-Agreement [WSAG05]. Figure 2-3 illustrates the interaction between the Application and the GMM, focusing on the main steps involved in this interaction. When a client issues a request, the application determines which services are required to fulfill it. These services represent either software executables (e.g. a mathematical algorithm) or computational resources. The application service translates these requirements to a WS-Agreement template, which is submitted to the GMM.

The middleware searches among the available service providers, which have registered their particular service specifications, such as contractual conditions, policies and QoS levels. When a suitable service provider is found, the application requirements are negotiated within the middleware by agents who act on behalf of the service providers as sellers and the applications as buyers. Once an agreement is reached between trading agents, a service

instance is created for the application, and a reference is returned to the application (which can subsequently be used to invoke it).

In the COllaborative VIrtual TEams (COVITE) prototype [COVITE04], suppliers and purchasers collaborate to procure supplies for a particular construction project by using the COVITE application. These projects are usually unique, very complex and involve many participants from a number of organizations. These participants need to work concurrently – thus requiring real time collaboration between geographically remote participants. Each consortium is in effect a Virtual Organization (VO). The application requires the search to take place across a large number of supplier databases to retrieve products matching a criteria set by the purchasers or contractors. The application enables a search to be performed using a cluster of machines in a Grid network.

The COVITE prototype application is divided into two functional services: Security Service and Multiple Database Search Services (MDSSs). Each MDSS enables searching across a large number of Supplier Databases (SD) using a Master Grid Service (MGS) instance via a cluster of machines in a Grid network. In this instance, the query is defined according to a data model that is specific to a given application domain. Arbitrary text queries (as in the Google search engine, for instance) are not allowed.

The COVITE application enables these VOs to plan, schedule, coordinate, and share components between designs, and from different suppliers. The ability of a free-market economy to adjudicate and satisfy the needs of VOs, in terms of services and resources, represents an important feature of the Catallaxy mechanism. Such VOs could require large amount of resources which can be obtained from computing systems connected over simple communication infrastructure such as the Internet. There are also possibilities for each of these VOs to try maximizing their own utility on the market.

4.1.2 Cat-COVITE prototype and the Catallactic Grid Markets

Different components of the COVITE application can be mapped to actors in a Catallactic market. Figure 4-1 shows the Cat-COVITE components and related Catallactic agents as buyers and seller in a service and a resource market.

The Complex Service, which is an abstract component consisting of the Master Grid Service (MGS), the Catallactic Access Point (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, 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 Complex Service includes the following activities:

- Translate a request to a Basic Service – of query job service type.
- Starts parallel negotiation with a number of agents representing Query Job Services (Basic Services).
- Sends a query to a list of Query Job Services (Basic Services).

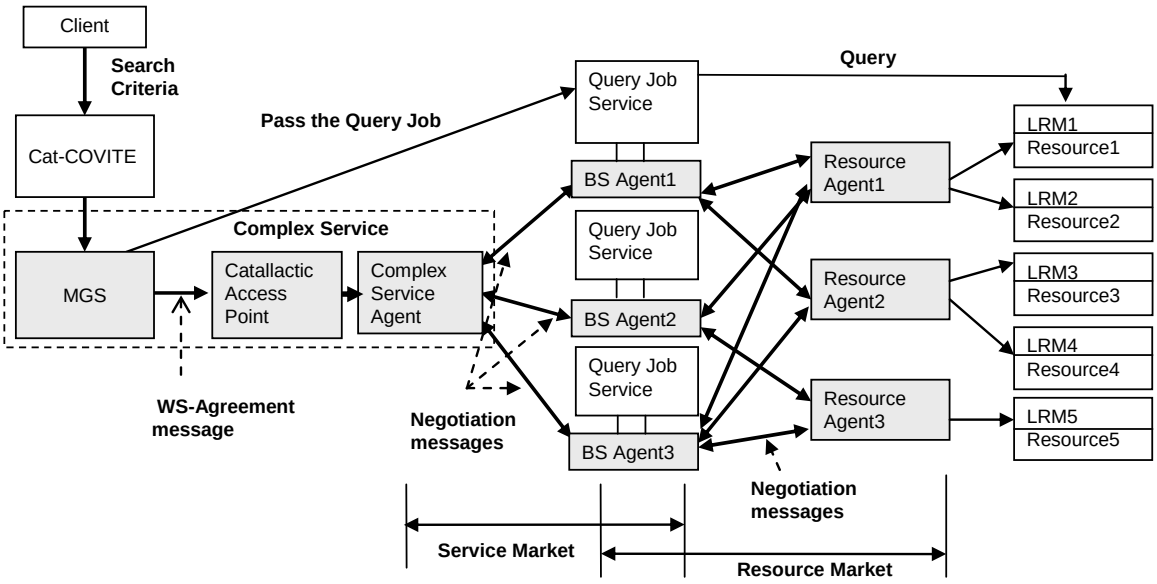

Figure 4-1 Cat-COVITE prototype and Catallactic Agents

The Query Job Service is the buyer entity in the resource market, and the LRMs 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).

The Query Job Service which is a type of basic service involves query execution on a particular database and consists of:

- Query Job Execution Environment (offers the deployment of "slaves", which are able to execute the query).
- Translation of query to resource requirements.

Within the Cat-COVITE application, the Query Job Service needs to support response time and the quality and quantity of the search. With this goal the Query Job Service buys resources in the resource market. Resource seller entities are able to provide a set of resources via the Local Resource Manager (LRM). The Resource Agents act on behalf of these LRMs, which hide the physical resources behind them.

Figure 4-2 shows 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. At the application layer, the application must provide an interface to the middleware which must issue the request for services to the middleware, and use the references to service instances provided by the middleware to execute such services.

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 instantiate the service on the execution environment selected during the negotiation process. Finally, at the Base Platform layer, a Resource is created to manage the allocation of resources to the service. 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).

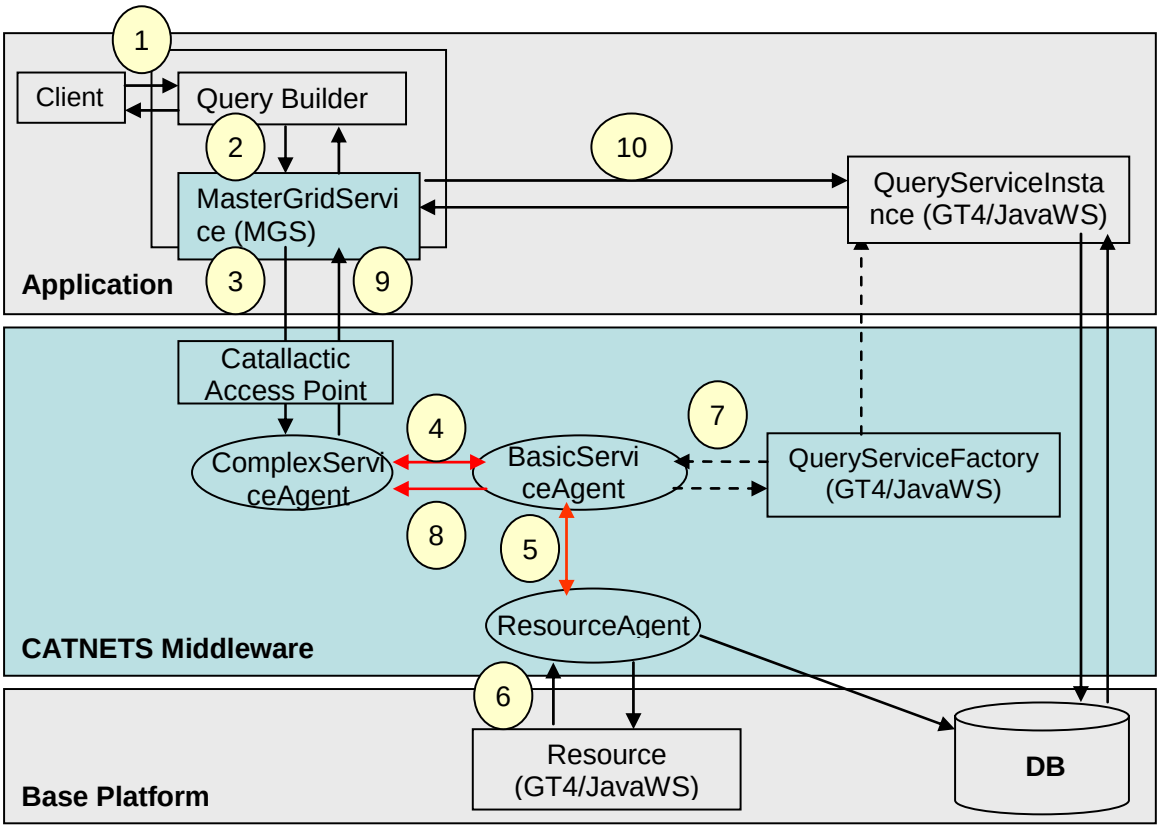

Figure 4-2 - Integration of Cat-COVITE and Catallactic Middleware

The flow of information among the logical components can be summarized as follows: a client issues a request to the application (1), which builds a query and requests the execution of query to the Master Grid Service (MGS) (2). The MGS contacts a Catallactic 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 (3). Note that the generator of the WS-Agreement ensures that there is an agreement on terms between the Agreement Initiator and the Agreement Offer provider. The creation of the agreement template in this way ensures that there is some consensus on the use of particular terms between the application and the CAP. The Complex Service Agent initiates Catallactic mechanisms to find the appropriate Basic Services and Resources. The Complex Service Agent uses discovery mechanisms implemented in the middleware Peer Agent Layer to locate Basic Service Agents providing a Query Service. When a number of Basic Service Agents are discovered, it starts negotiations with one of them (4). In turn such Basic Service Agent must discover and negotiate with a Resource Agent for query execution resources in the resource market (5). Negotiations are implemented by the Economic Framework Layer, where different protocols can be used depending on the agent's strategy. When an agreement with a Basic Service Agent is reached, the Resource Agent instantiate a Resource to keep track of the allocated resources and provides to the Basic Service Agent a handle for this resource (6). Subsequently, a Basic Service Agents use the Query Service Factory to instantiate the Query Service on the selected GT4 container (7). A Basic Service Agent returns to the Complex Service Agent the reference of the newly instantiated Query Service and the related resource(s) (8). The reference to the Query Service is returned to the MGS (9), which uses it to invoke the service, passing the query to be executed (10).

4.1.3 WS-Agreement in Cat-COVITE Prototype

The example scenario in terms of the Cat-COVITE application in the prototype interacting with the middleware is as follows: an MGS needs to run a search request. The MGS sends an Agreement Offer (AO), based on the Agreement Template (AT) downloaded from the Catallactic Access Point (CAP), to find a query job service. The Complex Service Agent, acting on behalf of the Complex Service (in this case represented by the MGS) negotiates with the Basic Service Agent for query services to fulfil the job. The agreement template (AT) specifies the service description elements that are allowed by the factory which advertises it.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:AgreementTemplate AgreementID="QueryTemplate-v001"
xmlns:wsag="http://schemas.ggf.org/graap/2005/09/ws-agreement">
<wsag:Name>QueryComplexService</wsag:Name>
<wsag:Context>
        <wsag:AgreementInitiator> <! -- can be a URI or a security identity of the
initiator --> NameOfTheInitiator </wsag:AgreementInitiator>
        <wsag:ExpirationTime>DateTime</wsag:ExpirationTime>
        <wsag:TemplateID>QueryTemplate-001</wsag:TemplateID>
        <wsag:TemplateName>QueryComplexService </wsag:TemplateName>
</wsag:Context>
<wsag:Terms>
        <BasicServiceType>QueryBasicService </BasicServiceType>
        <NumberOfBasicServiceNodes> <!-- between 1 to 10 -->
        </NumberOfBasicServiceNodes>
        <BasicServiceConstraints>
                <ResponseTimePerRequest>10
                <!-- maximum milliseconds -->
                </ResponseTimePerRequest>
         </BasicServiceConstraints>
        <Price> </Price>
</wsag:Terms>

</wsag:AgreementTemplate>
```

Listing 4-1 - Agreement Template

The agreement offer (AO) is initiated by the agreement initiator, in this case the MGS – and presented below.

If the agreement provider does not accept the offer, the agreement initiator has to send another agreement offer. The negotiation between the agreement provider and initiator, in this scenario, is based on price.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:AgreementTemplate AgreementID="QueryTemplate-v001"
xmlns:wsag="http://schemas.ggf.org/graap/2005/09/ws-agreement">
<wsag:Name>QueryComplexService</wsag:Name>
<wsag:Context>
        <wsag:AgreementInitiator> <! -- can be a URI or a security identity of the
initiator --> NameOfTheInitiator </wsag:AgreementInitiator>
        <wsag:ExpirationTime>DateTime</wsag:ExpirationTime>
        <wsag:TemplateID>QueryTemplate-001</wsag:TemplateID>
        <wsag:TemplateName>QueryComplexService </wsag:TemplateName>
</wsag:Context>
<wsag:Terms>
        <BasicServiceType>QueryBasicService </BasicServiceType>
        <NumberOfBasicServiceNodes>1 <!-- between 1 to 10 -->
        </NumberOfBasicServiceNodes>
        <BasicServiceConstraints>
                <ResponseTimePerRequest>10
                <!-- maximum milliseconds -->
                </ResponseTimePerRequest>
         </BasicServiceConstraints>
        <Price>100</Price>
</wsag:Terms>

</wsag:AgreementTemplate>
```

Listing 4-2 - Agreement Offer

4.2 Data Mining Services

An application instance based on data mining mechanisms is considered and discussed in this section. The basic problem addressed by the data mining process is one of mapping low-level data (which are typically too voluminous to understand) into other forms that might be more compact (for example, a short report), more abstract (for example, a descriptive approximation or model of the process that generated the data), or more useful (for example, a predictive model for estimating the value of future cases). At the core of the process is the application of specific data-mining methods for pattern discovery and extraction. This process is often structured into a discovery pipeline/workflow, involving access, integration and analysis of data from disparate sources, and to use data patterns and models generated through intermediate stages. A particular use of Grid computing in this context would be to combine services that implement specific phases in the discovery pipeline, which includes:

- Selection of a data set. The data set may be in a variety of different formats (such as Comma Separated Values, Attribute Relation File Format, etc) and a selection of attributes may also be required. This task is undertaken by the user.
- A converter may be necessary. Let Tc(ds) represent the time to achieve this conversion and any pre-processing required on the data – such as support for selection of attributes that are needed by the data set. This time is a function of the size of the data set (represented by variable ds). The task of conversion of the data set is hidden to the client and may be undertaken by the service provider.
- Selection of a data mining algorithm. This selection depends on the nature of the data and knowledge to be extracted from the data set. The selection process could be automated through the use of pre-defined rules, or based on the past experience of a user. This stage may be skipped if the user already knows which algorithm is required. Making this choice is often a difficult decision to make, and generally little support is provided in existing tools. A user is often presented with available algorithms, and has to make a choice

manually. Let Tas(n) represent the time to find a suitable algorithm from "n" possible algorithms. The following phases are hidden to the user and the complexity of them is handled by the application system.

- The fourth stage involves the selection of the resources on which the data mining algorithm needs to be executed. The choice is automated by the underlying resource management system via the Catallactic mechanism, if multiple instances of the same algorithm can be found. Let Trs(m) represent the time to find a suitable resource from "m" possible resources or resource bundles.
- The data mining algorithm is now executed - by checking security constraints on the remote resource, and often the data set may need to be migrated to the remote resource. Let $T^a_{md}(ds)$ represent the time for data migration, Tc represent the time to configure the service instance on a particular resource (such as checking of security credentials), and Te represent the time to execute the algorithm.
- The generated model from the data mining algorithm is now presented textually or graphically to the user. The model may now be verified through the use of a test set. Let Tpp(ds) represent the time to achieve this post processing on the data, and $T^b_{md}(ds)$ the time to migrate the data back to the client.

Hence, the total time required to undertake a particular data mining task is:

$$\tau = T_c(ds)+T_{as}(n)+T_{rs}(m)+ T^a_{md}(ds)+Tc + Te + Tpp(ds) + T^b_{md}(ds) \quad (4.1)$$

Assuming $T_c(ds) \approx T_{pp}(ds)$, $T^a_{md}(ds) \approx T^b_{md}(ds) = T_{md}(ds)$, and $T_c \ll T_e$, then $(T_c + T_e) \approx Te$ - hence we can simplify τ to:

$$\tau = 2(T_c(ds)+T_{md}(ds))+T_{as}(n)+T_{rs}(m)+T_e \quad (4.2)$$

The use of the Catallactic market is primarily intended to associate a cost with Tc(ds), Tmd(ds) and Te. A service provider that has a high bandwidth is able to provide a fast pre- and post- processing service. Similarly, having a low execution time for the analysis algorithm would have the highest cost in the market. The times Tas(n) and Trs(m) are associated with the service and resource markets respectively. The Catallactic middleware is intended to mimic the behaviour of a Catallactic market, and needs to be efficient enough to be able to reduce these times compared to other market mechanisms.

We have assumed that pre-processing of data is undertaken by the client and the post-processing tasks by the specialized service providers. Assuming that there are "k" services in a pipeline, with each stage being undertaken by different service and resource providers (representing data pre-processing and transformation, analysis, post-processing and visualization) we can represent, as an upper bound, a time of (k x τ) to represent the total time for the data mining task. It would now be necessary to engage the Catallactic middleware "k" times to find suitable service instances in the market.

4.2.1 Cat-Data Mining prototype

To demonstrate our ideas, a proof-of-concept prototype has been developed and named Catallaxy Data Mining (Cat-DataMining), which extends the implementation in the COllaborative Virtual TEams (COVITE) [COVITE04] project. As previously described, the Cat-COVITE is based on a Service Oriented Architecture, and consists of three main

elements: (i) one or more user services; (ii) a "Master Grid Service" (MGS) and (iii) one or more service instances that are being hosted on a particular resource. Cat-COVITE currently supports searching through distributed product catalogues (each being a database wrapped as a Web Service), achieved by launching multiple concurrent searches. This database search has been extended with data analysis services which may operate on data sent by a client service, or data already available where the analysis service is hosted.

Previous work has involved translating data mining algorithms supported in the WEKA toolkit into Web Services - a number of classification and clustering algorithms have been converted [ALI05]. The Catallaxy Data Mining (Cat-DataMining) prototype makes use of classifier services that implement a J48 decision tree classifier, based on the C4.5 algorithm [QUINLAN93].

The J48 service has two options: classify, and classify graph. The classify option is used to apply the J48 algorithm to a data set specified by the user. The data set must be in the ARFF format, which essentially involves a description of a list of data instances sharing a set of attributes.

The result of invoking the classify operation is a textual output specifying the classification decision tree. The classify graph option is similar to the classify option, but the result is a graphical representation of the decision tree created by the J48 service. A number of other visualization services are also supported. We therefore have a market of J48 services, each being hosted on different resources.

4.2.2 Data Mining Services and Grid Markets Middleware (GMM)

Figure 4-2and Figure 4-3 show the prototype components and related Catallactic agents as buyers and seller in the Grid service market and Grid resource market: case (a) details one service invocation, while case (b) details a service workflow invocation. The prototype is composed of four main components: the Master Grid Service (MGS) – a Complex Service requestor, the Catallactic Access Point (CAP), the Data Mining Services - as types of Complex and Basic Services, and job execution resources - as computational resources. The Complex Service agent acting on the request of MGS is the buyer entity in the service market, and the Basic Service is the seller entity in the service market.

The Basic Service involves data mining job execution and consists of a data mining Job Execution Environment, which offers the deployment of multiple "slaves" able to execute the data mining task. Within the Cat-DatMining prototype, the data mining Basic Service needs to be able to provide a response time as an important characteristic - this is equivalent to a sum of parameters Tc(ds) + Tmd(ds) + Te from equation 4.1 in Section 4.2. With this goal the data mining Basic Service buys resources in the resource market. Resource seller entities are able to provide a set of resources via the Local Resource Manager (LRM). The Resource Agents act on behalf of these Local Resource Managers (LRMs), and provide an interface to the physical resources they manage.

The data mining Basic Service is the buyer entity in the resource market, and the Local Resource Managers are the seller entity on the resource market. The main functionalities of Basic Service agent at the resource market are: (i) co-allocation of resources (resource bundles) by parallel negotiation with different resource providers (local resource manager entities); (ii) informing the Complex Service about the outcome of resource negotiation.

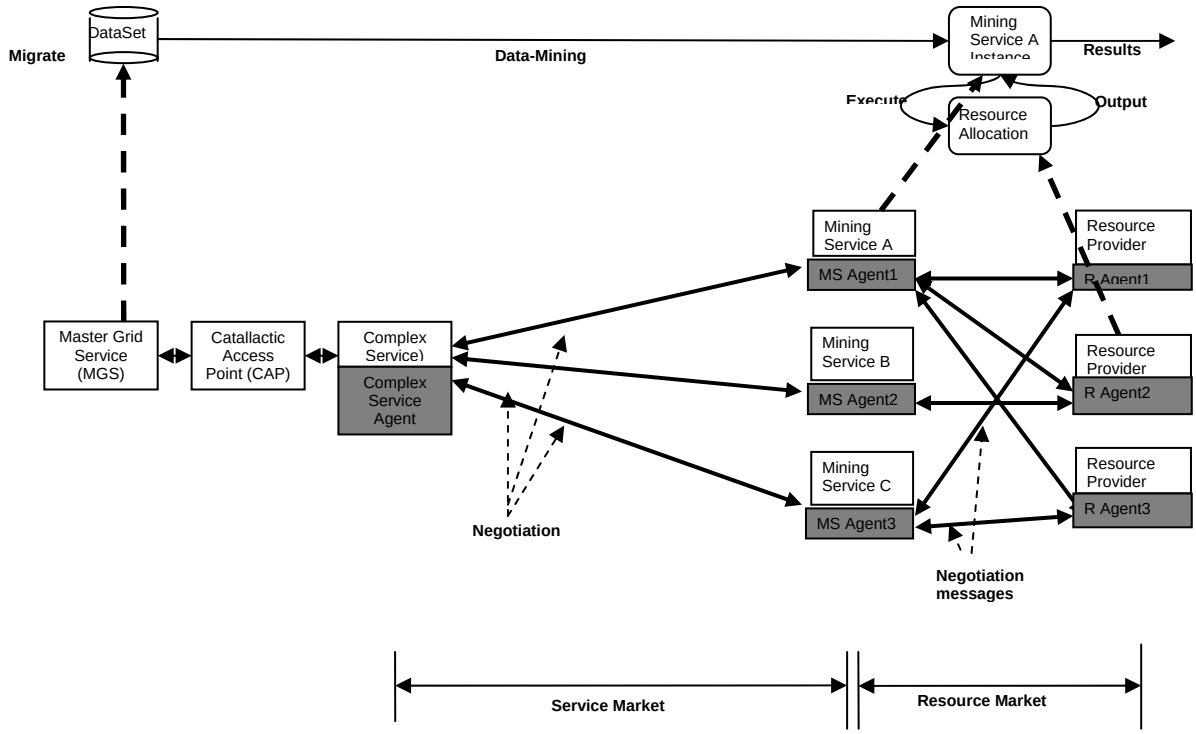

Figure 4-3 - Cat-DataMining prototype – the Catallactic agents and the markets – one service invocation

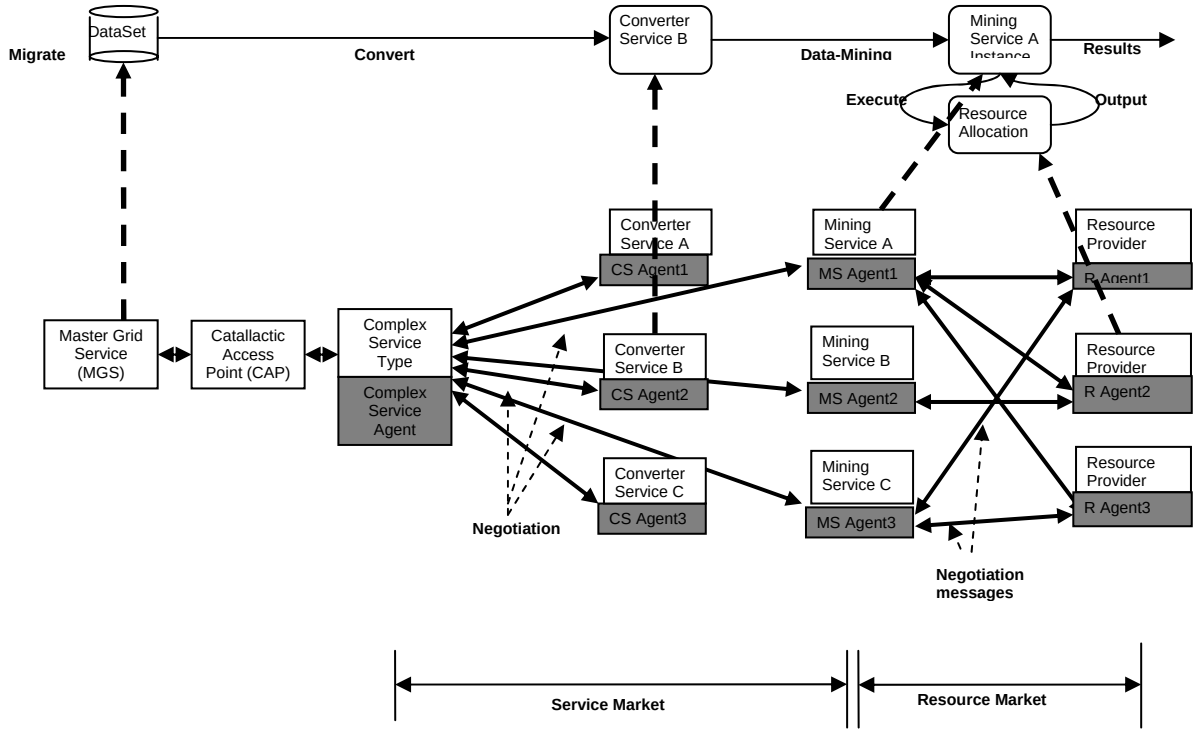

Figure 4-4– Cat-DataMining prototype – the Catallactic agents and the markets – workflow of services invocation

The data mining scenario of the Cat-DataMining prototype involves an MGS which needs to run a data mining job. The MGS sends an AgreementOffer (AO), based on the AgreementTemplate (AT) downloaded from the CatallacticAccessPoint (CAP), to the CAP to find a data mining service. The CAP is a Web Service located on a machine providing a Catallactic market access point, acts as a WS-Agreement provider for the application and as a

factory for Complex Service Agents which will start the negotiation process. The CAP therefore provides an entry point into the market, and can allow existing Grid applications to make requests directly to it. The Complex Service Agent, acting on behalf of the MGS (as a complex service) chosen by the CAP, negotiates with the Basic Service Agents (in the Cat-DataMining markets environment) for data mining services. The AT specifies the service properties that are necessary to create an instance of a service using a factory service. The AT and AO are provided in the Appendix. The DecisionMaker, which is part of the CAP, takes the decision of accepting or rejecting the agreement offer sent by the MGS. There are multiple factors involved in this decision, such as: the parameters in the agreement offer that are part of the agreement template, and the possibility of finding the available Basic Service(s) within the Catallactic market at the budget specified by the MGS as service requestor.

Figure 4-5 shows the placement of logical components along the three layers: the application layer, the Catallactic middleware layer and the base platform layer – in the use case of one service invocation (see Figure 4-3).

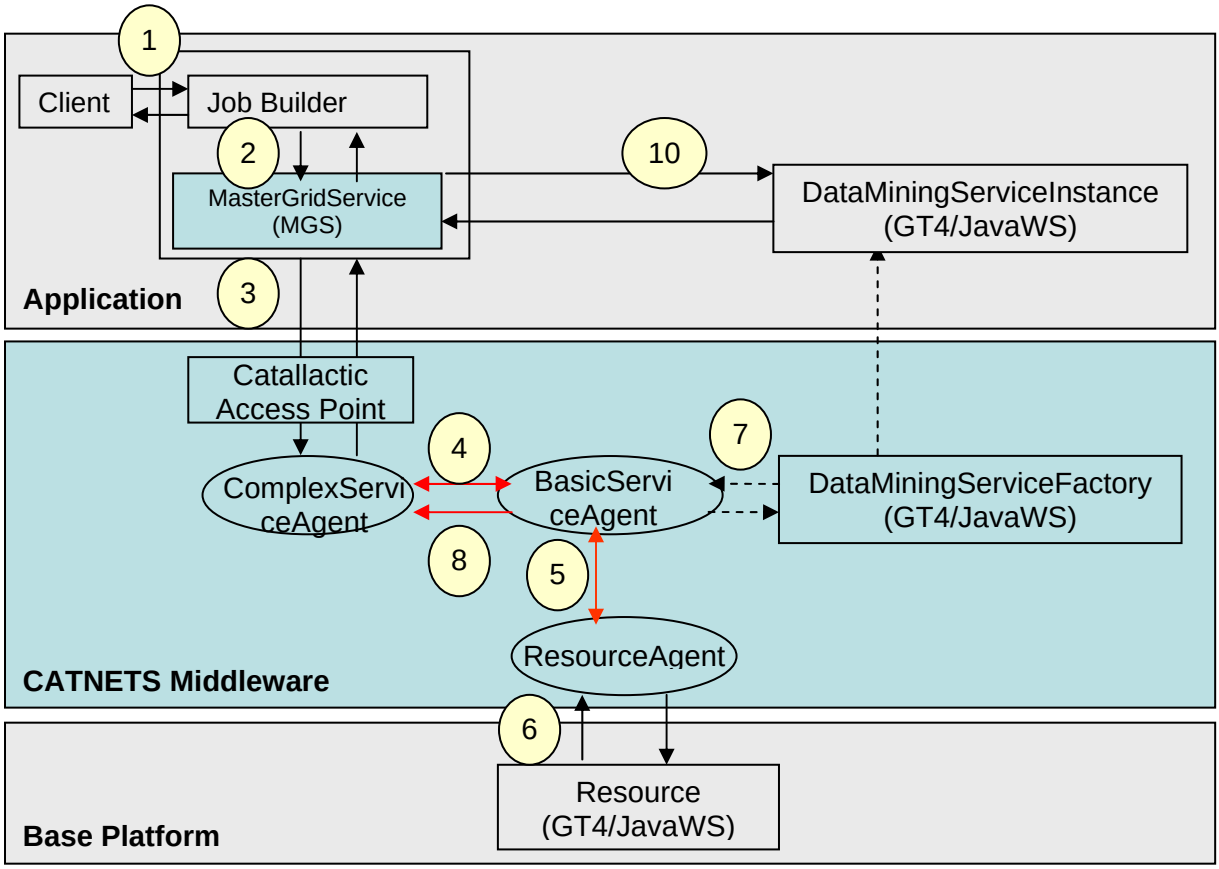

Figure 4-5 – Interaction between prototype, Catallactic middleware and computational resources

At the application layer, an interface must be provided to issue the requests for services to the middleware, and use the references to service instances provided in response. At the middleware layer, a set of agents provide the capability to negotiate for services and resources. The Complex Service agent acts on behalf of the application and initiates the negotiation. Basic Service and Resource agents manage the negotiation for services and resources respectively. A Service Factory is provided to instantiate the service on the hosting platform selected during the negotiation process. Finally, at the Base Platform layer, a Resource is created to manage the allocation of resources to the service. This resource represents the "state" of the service from the perspective of the middleware (however this does not mean the service is stateful from the perspective of the application). The flow of

information among the logical components can be summarized as follows: a Client issues a request to the application (1), which builds a data mining job and requests the execution of this job to the MGS (2). The MGS contacts a CAP asking for a WS-Agreement template for such a service. The MGS fills in the template and sends back an AO (3). The Complex Service Agent initiates the Catallactic mechanism to find appropriate Basic Services and Resources. The Complex Service Agent uses the discovery mechanisms implemented in the middleware to locate Basic Service Agents providing the J48 Service. When a number of Basic Service Agents are discovered, it starts negotiations with one of them (4). In turn such Basic Service Agent must discover and negotiate with a Resource Agent for resources (5). Negotiations are implemented by the Economic Framework Layer, where different protocols can be used depending on the agent's strategy. When an agreement with a Basic Service Agent is reached, the Resource Agent instantiate a Resource to keep track of the allocated resources and returns to the Basic Service Agent a handle for this resource (6). Consequently Basic Service Agents use the service Factory to instantiate the data mining service on the selected GT4 container (7). Basic Service Agent returns to the Complex Service Agent the End Point Reference (EPR) to this data mining service instance (8), forwarded to the MSG (9), which uses it to invoke the service (10). The CAP therefore provides an interface between an application wishing to discover suitable services, and the underlying resources that host those services.

4.2.3 WS-Agreement in Cat-DataMining Prototype

WS-Agreement is used in Cat-DataMining, and 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. When using WS-Agreement in our prototype, several parts need to be specified [WSAG05]: agreement name, the agreement context - parties to the agreement, reference to the service(s) provided in support of the agreement, and the lifetime of the agreement. 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. Finally, guarantee terms which specify the service levels that the parties are agreeing to. An example of an Agreement Template and Offer used by Cat-DataMining is provided below.

```
<AgreementTemplateLite>
        <Name>DataMiningComplexService</Name>
        <Context>
                <AgreementInitiator>Cardiff-A
                </AgreementInitiator>
                <StartingTime>2005-12-12T13:00:00
                </StartingTime>
                <TerminationTime>2005-12-12T14:00:00
                </TerminationTime>
        </Context>
        <Terms>
                <Basic ServiceType>
                </BasicServiceType>
                <NumberOfBasicServiceNodes>
                <!-- between 1 to 10 -->
                </NumberOfBasicServiceNodes>
                <BasicServiceConstraints>
                        <ResponseTimePerRequest>10
                        <!-- maximum milliseconds -->
                        </ResponseTimePerRequest>
                </BasicServiceConstraints>
                <PayForService>
                </PayForService>
        </Terms>
</AgreementTemplateLite>
```

Listing 4-3 - Agreement Template

The AO is initiated by the agreement initiator (the MGS) in this case. An AO is as follows:

```
<AgreementOfferLite>
        <Name>DataMiningComplexService</Name>
        <Context>
                <AgreementInitiator>Cardiff-A
                </AgreementInitiator>
                <StartingTime>2005-12-12T13:00:00
                </StartingTime>
                <TerminationTime>2005-12-12T15:00:00
                </TerminationTime>
        </Context>
        <Terms>
                <BasicServiceType>J48Service
                </BasicServiceType>
                <NumberOfBasicServiceNodes>1
                </NumberOfBasicServiceNodes>
                <BasicServiceConstraints>
                        <ResponseTimePerRequest>10
                        </ResponseTimePerRequest>
                </BasicServiceConstraints>
                <PayForService>100
                </PayForService>
        </Terms>
</AgreementOfferLite>
```

Listing 4-4 - Agreement Offer

4.3 Generalising Application Scenarios

The application scenarios we consider are centered on the concept of "Application Layer Networks" (ALN), which includes generic application classes such as Content Distribution Networks (CDN), Peer-to-Peer Networks (P2P) or Grid. An ALN is a collective entity that provides a certain service using a composition of service elements that cooperate among each other, organized as an overlay network.

Based on these definitions, we identify existing systems that may be suitable candidates for ALNs. Three such scenarios include: Grids (Planetlab), CDN (Coral) and P2P file sharing (BitTorrent).

PlanetLab, as described in [BAVIER04] is a geographically distributed overlay network designed to support the deployment and evaluation of planetary-scale network services. Two high-level goals shape its design. First, to enable a large research community to share the infrastructure, PlanetLab provides distributed virtualization, whereby each service runs in an isolated slice of PlanetLab's global resources. Second, to support competition among multiple network services, PlanetLab decouples the operating system running on each node from the network-wide services that define PlanetLab, a principle referred to as "unbundled management".

CoralCDN [FREEDMAN04] is a decentralized, self-organizing, peer-to-peer web-content distribution network that leverages the aggregate bandwidth of volunteers running the software. CoralCDN uses DNS redirection to transparently redirect a Web browser to nearby Coral web caches. These caches cooperate to transfer data from nearby peers whenever possible, minimizing both the load on the original Web server and the end-to-end latency experienced by browsers. This requires two mechanisms: finding a close enough peer, and finding a close copy of the requested object. The first is achieved by mapping Coral servers and clients in clusters based on latency. The second is done using a locality-aware request routing algorithm.

BitTorrent [COHEN03] is a peer-to-peer network protocol that allows multiple people to download the same file at the same time. Using the protocol, users may also upload pieces of the file to each other. This redistributes the cost of upload to downloaders, where it is often not even metered, thus making hosting a file with a potentially unlimited number of downloaders affordable. BitTorrent uses a central component, called tracker, which is not involved in the actual distribution of the file; instead, it keeps meta-information about the peers that are currently active and acts as a rendezvous point for all the clients of the torrent.

4.3.1 Application requirements for Catallaxy - generalization

In general, different types of applications and users in a Grid form a Virtual Organization (VO) for planning, scheduling, and coordination phases within specific projects or businesses. A Grid infrastructure allows the users of a VO to interact among them for the duration of that particular VO. The ability of market based coordination is to adjudicate and satisfy the needs of VOs, in terms of policy adherence and QoS levels. Such VOs could require large amounts of resources, which may be obtained from computing systems connected over simple communication infrastructure such as the Internet. There could also be possibilities for these VOs to try to maximize their own utilities on the market. There are many requirements upon such a coordination mechanism, i.e. applications and users may have different requirements upon auction mechanisms and its underlying institutional rules.

A generalisation of applications represented via a detailed UML sequence diagram is shown in Figure 4-6. The application requires allocation of services and resources which are not

necessarily located on-site; therefore a request has to be made to an access point/middleware in which negotiations take place to finding and allocating them.

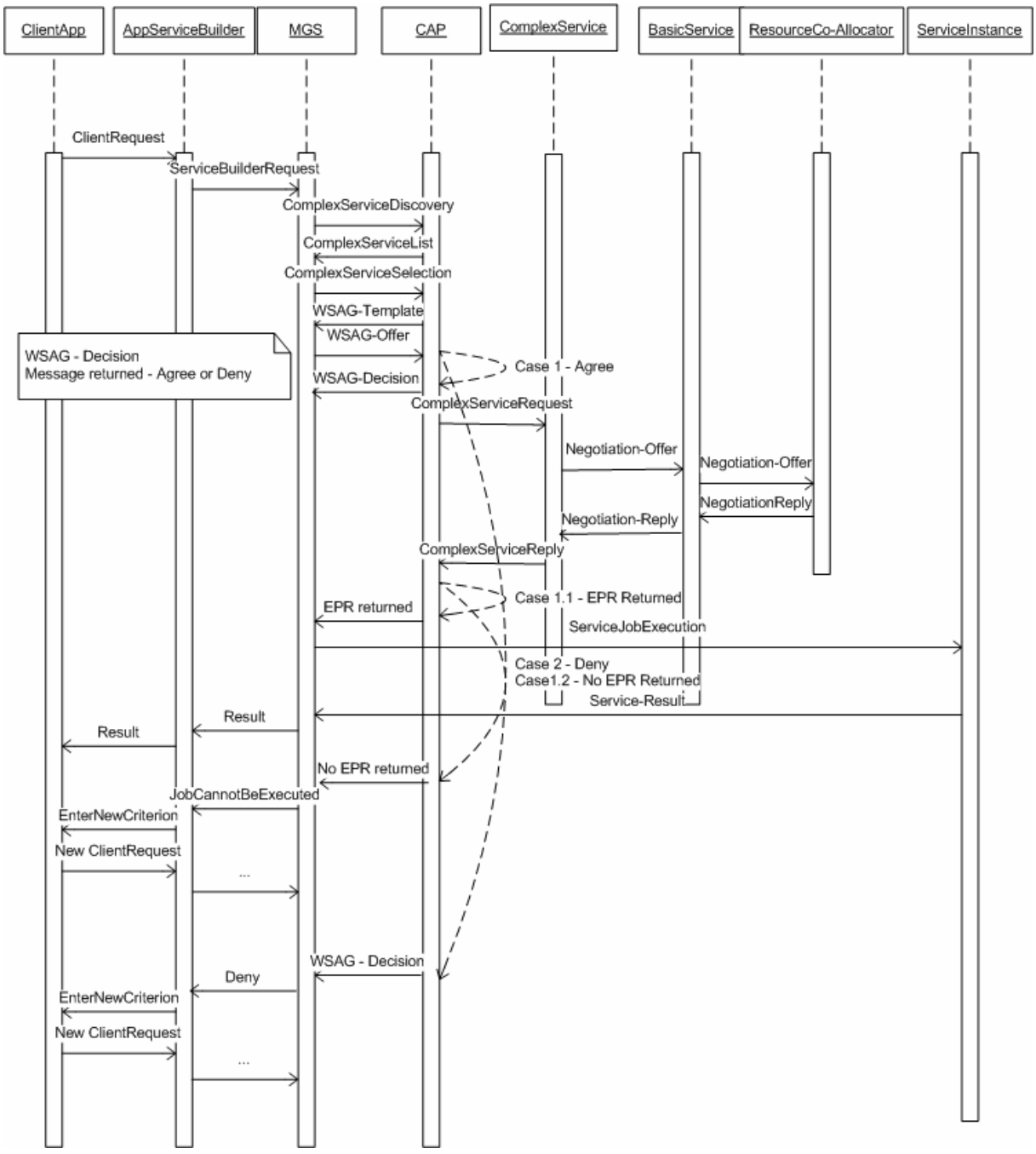

Figure 4-6- Generalisation of application – UML Sequence Diagram

There are two modules in our design that play an important role in generalising the application requirements for Catallaxy mechanism:

- Firstly, the Master Grid Service (MGS) module – the MGS is the module interface between the every application modules and Catallaxy markets. MGS could be seen as a service interface that could be implemented by any type of application in order to benefit from and connect to Catallaxy markets. This module is generic and is generalized for any type of application and requires some application specific logic parameters, as for

example the budget and the service requirement from the user. The MGS is connected to the CAP.

- Secondly, the Catallactic Access Point (CAP) – the CAP is the second module interface of Application (generic) and Grid Market Middleware (GMM). CAP is a web service interface that is exposed to every application via the MGS. The functionality of CAP is to provide an entry point for connection to the GMM, to list the complex services available on a specified market, as well as to exchange the agreements (template/offer) between the GMM and application.

Two methods are very important and available via the CAP:

- getAgreementTemplate() – this method provides access to the agreement template hosted by the CAP repository (in the MySQL format).
- receivedAgreementOffer() – this method is used to send the agreement offer with parameters specific to the Catallaxy market.

A view of actors involved in a generalized application prototype is shown in Figure 4-7. 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.

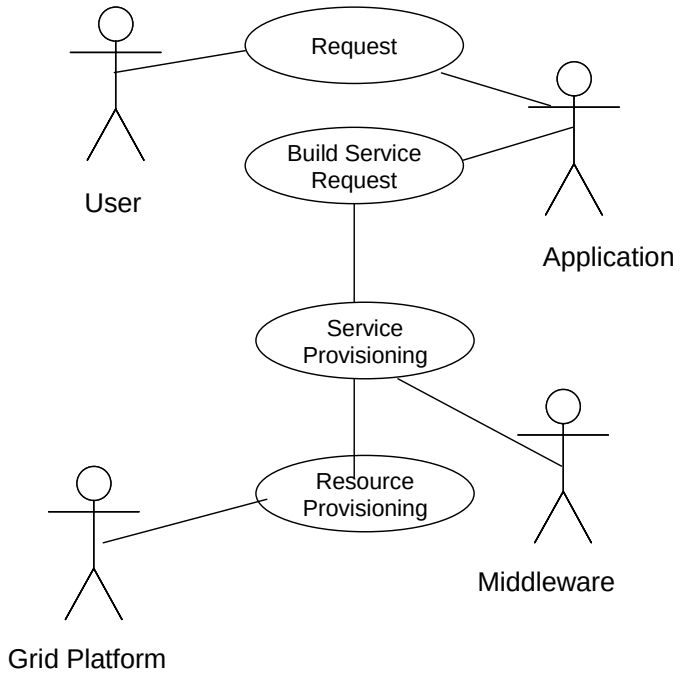

Figure 4-7 Generalisation of application – UML Use Case Diagram

Figure 4-8 shows the detailed components and repositories of an application that access the Catallactic middleware, the components and the modules of interaction between application and middleware, as well as the workflow among the components. The repositories could be any type of database (MS SQL, MySQL, Oracle, etc) or files systems. The CAP_WS is a wrapper in the format of a Web Service which provides the location of a CAP closer to the user location. There are different ways of "enforcing" or "finding" the user location, such as: user's digital certificate, in which the location is specified as a parameter in the certificate, "ping" time between the user and multiple available CAPs, etc.

Application settings, repositories, and details about two scenarios are presented above in section 4.1 and 4.2. These involve the use of a Query Service and a Data Mining Service, additional details are provided in Annex B.

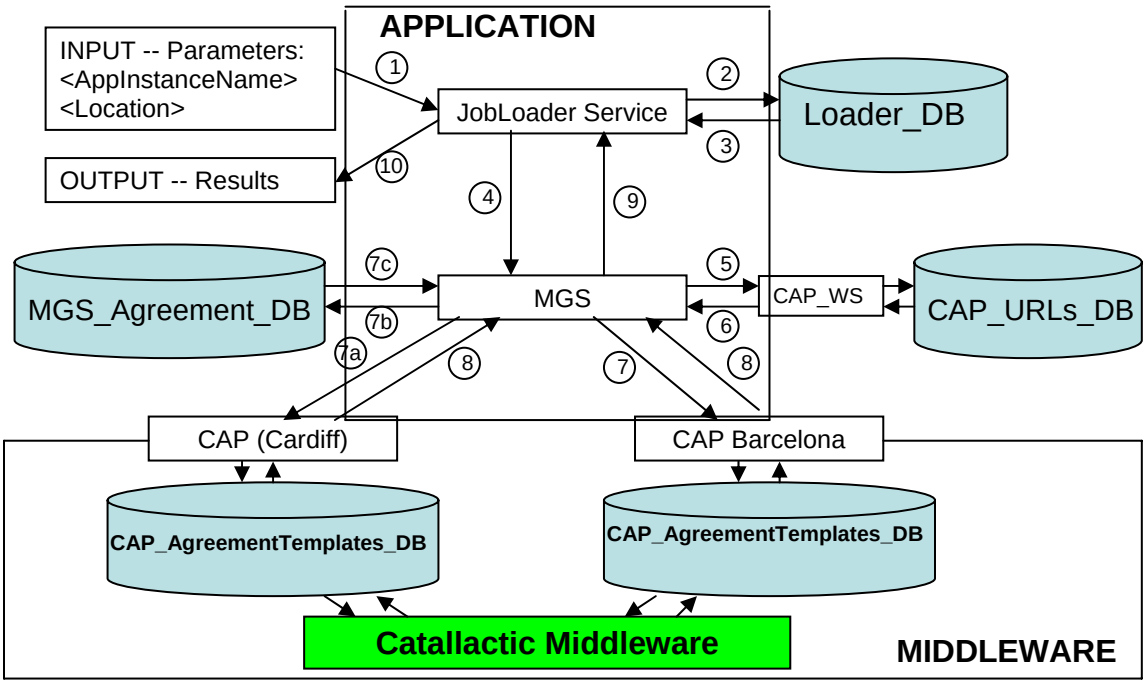

Figure 4-8. **Generalisation of application**

The workflow process, in a generalised application that connects to a Catallactic GMM is shown in the Figures 4-6, 4-7 and 4-8 above, and could be summarised as:

1 – A user enters the request criterion.

2 – A type of Job Loader Service interprets the user request and transforms it into a service request.

3 – A generic service, interpreted by Master Grid Service in the above diagrams, requires an application specific logic, such as: start a workflow of services to fulfil the user request, or obtain a budget. Any other parameters could be customised and added to this module.

4 – The Catallactic Access Point lists services available on the market (as complex or basic services) and establishes the necessary agreements.

5 Middleware Implementation

5.1 Implementation toolkits

5.1.1 Middleware implementation toolkits

The middleware toolkits selection process was carried out taking into account three different but related aspects, which are potential application scenarios, software architecture, and the evaluation of a number of middleware toolkits. For a detailed evaluation, see WP3 year 1 deliverable [WP305]

Concerning the evaluation, six toolkits were selected and reviewed: DIET and JADE agent platforms, J2SE, WSRF/OGSA, Web Services and JXTA. The evaluation includes their functional properties according to the software architecture defined for CATNETs, their technical characteristics and their suitability as a development toolkit.

A condensed view of all requirements, in functional, technical and development views is obtained considering the following criteria:

- Modularity to achieve architectural flexibility required to implement the Catallactic middleware into different platforms and using diverse middleware toolkits.
- Amenability: The middleware toolkit should be able to cover as much as possible of the ALN domains, like Grid, P2P and CDN.
- Performance & Scalability: The middleware toolkit should allow the organization of a huge number of software agents in a decentralized way, and their interactions.
- Completeness: The set of functionalities provided by the middleware toolkit should allow covering as much as possible of the desired requirements of the P2P Agent Layer).
- Development: In order to support the CATNETS middleware development, the middleware toolkits should provide be mature, have a rich set of development tools and good documentation.

Figure 5-1 shows an illustration of this unified view. Each of the pentagon axis represents one of the criteria. For each criteria the middleware toolkits best covering it is indicated.

It can be seen in the previous figure that CATNETS middleware will comprise a combination of different middleware toolkits, like DIET with JXTA and WSRF/OGSA, which achieve a good balance between the functional and non functional requirements.

Tests were carried out on middleware toolkits to confirm the feasibility of this composition, in the sense that we could integrate JXTA Discovery with the DIET Agents platform. Also the invocation of Grid Services from Java applications using the Java Globus API has been tested.

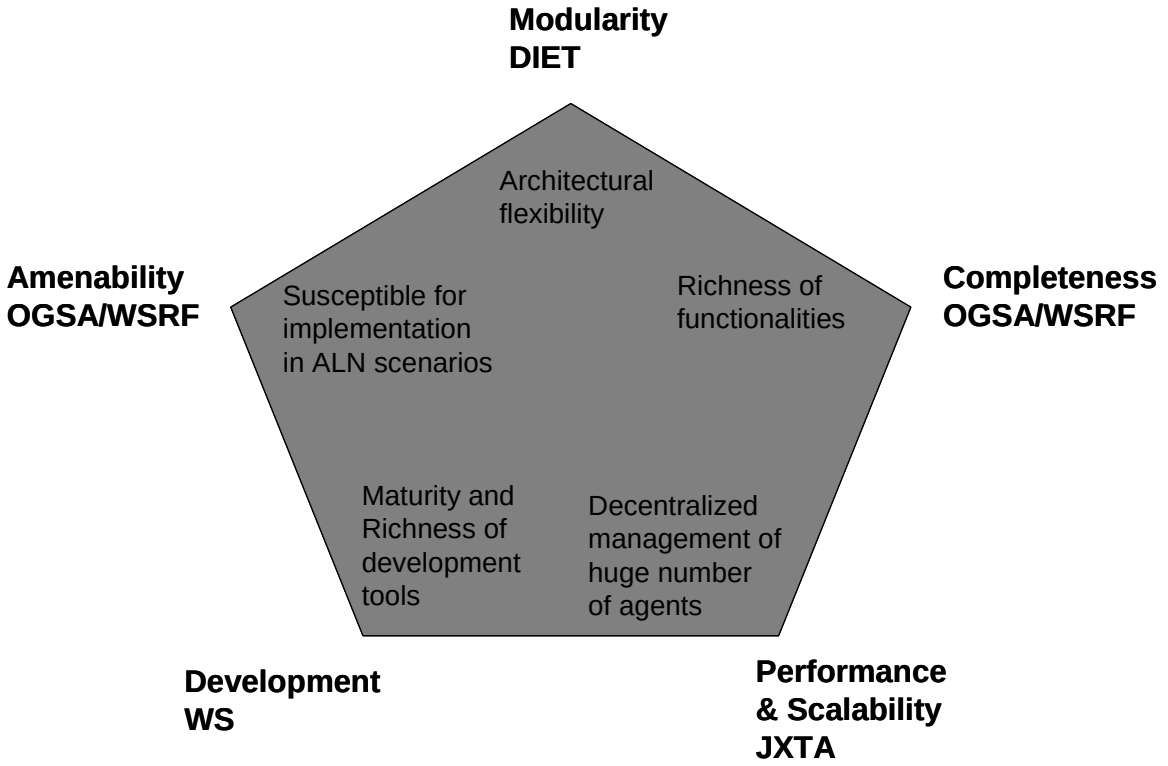

Figure 5-1 - Condensed view of the key middleware toolkit evaluation criteria

5.1.2 Development toolkits

Provided the number of different toolkits we are using in CATNETS middleware implementation, we decided to integrate all development into a single, extensible tool which provide us with rich tools for development/integration. Concerning Java development, Eclipse platform [ECLIPSE06] is the state of the art tool for this purpose, and it is presently supported by most of the major software development companies as well as numerous open source projects.

The Eclipse Platform's principal role is to provide mechanisms to seamlessly-integrate development tools. These mechanisms are exposed via well-defined API interfaces, classes, and methods. The Platform also provides useful building blocks and frameworks that facilitate developing new tools.

A plug-in is the smallest unit of Eclipse Platform function that can be developed and delivered separately. Usually a small tool is written as a single plug-in, whereas a complex tool has its functionality split across several plug-ins. Except for a small kernel known as the Platform Runtime, all of the Eclipse Platform's functionality is located in plug-ins. The JDT adds the capabilities of a full-featured Java IDE to the Eclipse Platform

Currently, there are numerous plug-in available, either developed under the Eclipse project's umbrella or by third parties, both commercial and open source. Among them, we have found some which are relevant to the prototype objectives

At the present, we are using the Web Standard Tools (WST) plug-in [WST06], which provides a handy tool to work with Web Services in the scope of the CATNETS project. Web

Services are manipulated in several CATNETS scenarios and the CAP itself is exposed as a Web Service.

WST provides, between others, the following functionalities:
- Develop XSD and XSLT for XML based Web pages and Web services
- Develop and publish WSDL schema on UDDI registries
- Explore UDDI registries and dynamically test Web services via WSDL
- Test Web services for WS-I compliance

In addition, we are using plug-ins provided for third parties for UML modeling as a primary design and documentation tool for the middleware prototype.

We are also evaluating the integration of the Test and Performance Tools to enhance the run time monitoring of middleware components, as well as the trace of application events. Finally, we do not discard the option to develop our own plug-in for such functions as middleware deployment and operation (launch, stop, change configuration parameters, etc).

5.1.3 Testing toolkits

Due to the complexity of the CATNETS middleware and the inherent unpredictability characterized open systems like the internet and P2P systems (both scenarios present in CATNETS), it is required a rich tool for both load and functional testing of the middleware under different environmental conditions.

Apache JMeter [JMETER06] is a tool designed to load test functional behaviour and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions, including the utilization of custom test classes (called “test samplers”). It can be used to simulate load on a server, network or object to test its strength or to analyze overall performance under different load types. Some features make particularly suitable to the prototype’s objectives:

- Full multithreading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- GUI interface allows faster definition and operation of test plans.
- Highly extensible by means of pluggable Samplers allow unlimited testing capabilities.
- Several load statistics may be chosen with pluggable timers.
- Data analysis and visualization plug-ins allow great extendibility as well as personalization.
- Functions (which include JavaScript) can be used to provide dynamic input to a test
- Scriptable Samplers to build complex test logic.

Description of JMeter TestPlan components:

A test plan describes a series of steps that JMeter will execute when run. A complete test plan will consist of one or more Thread Groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements.

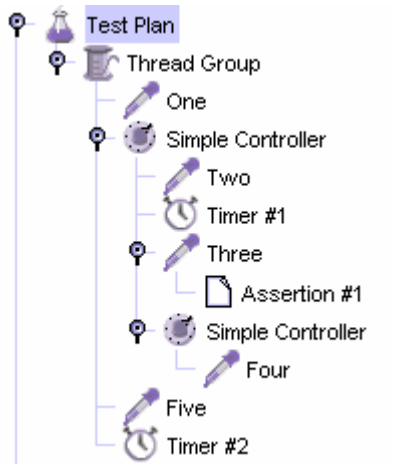

Figure 5-2 Structure of a test plan

In Figure 5-2 a test plan composed of several building blocks is showed. JMeter User is represented as a Thread Group. Each User runs one or more threads to perform several operation on the test setup. The basic operation is the TestSampler. In this figure we see several of these TestSamplers, named "One", "Two", etc. In order to coordinate several TestSamplers, different logical controllers can be used, in the example "Simple Controler" is used. Timers and assertions to post process data can be also added into the TestPlan.

Thread group elements are the beginning points of any test plan. All elements of a test plan must be under a thread group (Figure 5-3). As the name implies, the thread group element controls the number of threads JMeter will use to execute your test. The controls for a thread group allow you to:

- Set the number of threads
- Set the ramp-up period
- Set the number of times to execute the test

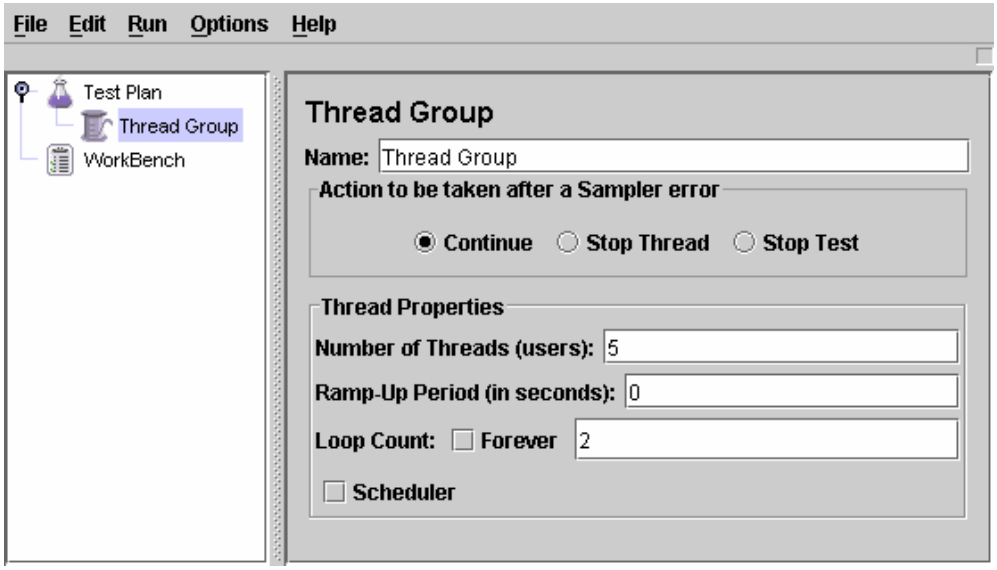

Figure 5-3 – Definition of a test plan

Each thread will execute the test plan in its entirety and completely independently of other test threads. Multiple threads are used to simulate concurrent connections to the server application.

Controllers let you modify the ordering and flow of a JMeter test script. Controllers may have as child elements any of the following: Samplers (requests), Configuration Elements, and other Logic Controllers. Controllers can change the order of requests coming from their child elements.

Samplers tell JMeter to send requests to a serve (Figure 5-4). JMeter currently incorporates several predefined samplers: FTP Request, HTTP Request, JDBC Request, Java object request, LDAP Request, SOAP/XML-RPC Request and Web Service Request.

Each sampler has several properties you can set. You can further customize a sampler by adding one or more Configuration Elements to it. If you are interested in having JMeter perform basic validation on the response of your request, add an Assertion to the Request controller. Add a Listener to your Thread Group to view and/or store the results of your requests to disk.

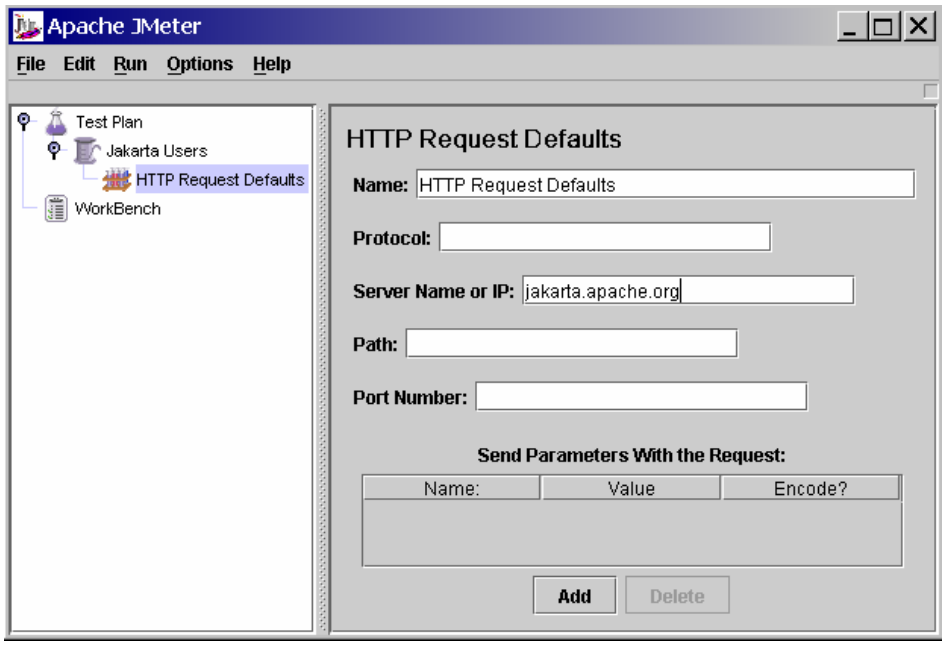

Figure 5-4 – Definition of an HTTP request

In addition to the predefined request, the Java object request can be extended in order to build user defined TestSamplers. This brings the possibility to develop custom tester classes. The Java Sampler lets you control a java class that implements the JavaSamplerClient interface. By writing your own implementation of this interface, you can use JMeter to harness multiple threads, input parameter control, and data collection. In the scope of CATNETS project we have developed such JavaSamplerClient implementation which allows us controlling several tests on the middleware. Details on these functional tests are covered in section 5.4 of this document.

5.2 Detail component implementation

5.2.1 Middleware implementation

The middleware was implemented as a set of simple, specialized DIET agents. The details on the interaction of these agents are organized in a way that resembles the separation of concerns proposed by the architecture, as is shown in Figure 5-5. Framework agents supports the basic functions needed to implement economic algorithms, like access to markets. Peer Agent Layer agents implement the low level functionalities to support system execution. Service Provider agents interface with the implementation platforms (JXTA and GT4).

The Overlay Network, Object Discovery and Communication functions where implemented using JXTA Peer Resolver Protocol in a network of Rendezvous Peers that uses a DHT to maintain and route messages among nodes. The management of local resources, in this case services offered by the service providers, uses the WSRF framework offered by GT4.

In the proposed implementation, Trading Agents implementing economic algorithms share the supporting agents from lower levels. This separation allows for scalability as the number of supporting agents can be dynamically adapted to workload. This separation also offers location transparency for agents.

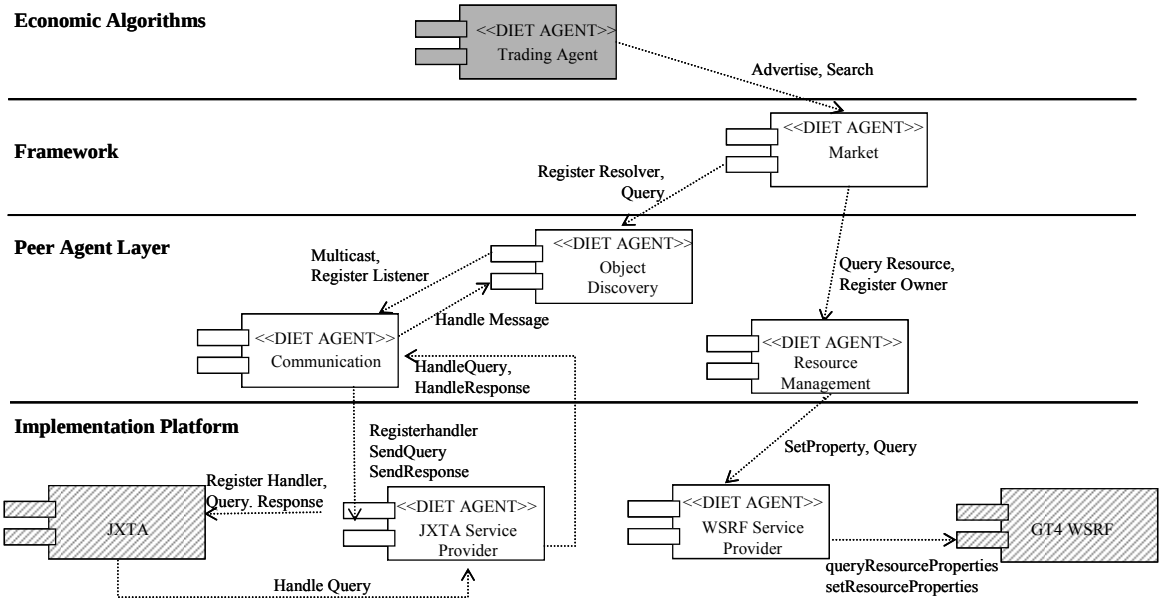

Figure 5-5 - Organisation of middleware components

More details on the middleware implementation are covered in the WP3 year 1 deliverable [WP305]

5.2.2 Middleware API

The middleware implements a simple, yet flexible and feature rich programming interface to facilitate the development of agent based applications that can interact over P2P networks to trade for resources and services. This API turns around two main concepts: P2pAgentHosting and P2PAgent.

A P2pAgentHosting offers the runtime support to create and execute agents. It must be initialized by the application with the required configuration information and then can be used

to create P2pAgents to perform application specific actions, passing agent specific parameters.

Any agent in the environment must extend the P2pAgent class, which provides the basic functionalities to interact with the environment and with other agents. Agents executes asynchronously in their own threads, which are managed by the P2pAgentHosting. Classes implementing agents must follow an event oriented model, where the agent's main logic is not embodied in a "main" method, but spread in different methods on which it responds to external events, such as the reception o messages from other agents.

An agent should extend some abstract methods of this class to implement its own logic:

- Run: agent main logic. It is invoked just once at agent creation, but can be called from other agent's methods (see below)
- handleMessage: handle the reception of a message from another agent
- handleEvent: handle the expiration of an event scheduled at a specific time
- handleGroupcast: handle the reception of a message sent to a group the agent belongs to an agent

When the agent is created, the application receives an AgentStatus object, which allows it to check if the agent has terminated and return any results. It is important to notice that once the agent is created, the application should not directly communicate with it.

This programming model is summarized in Figure 5-6.

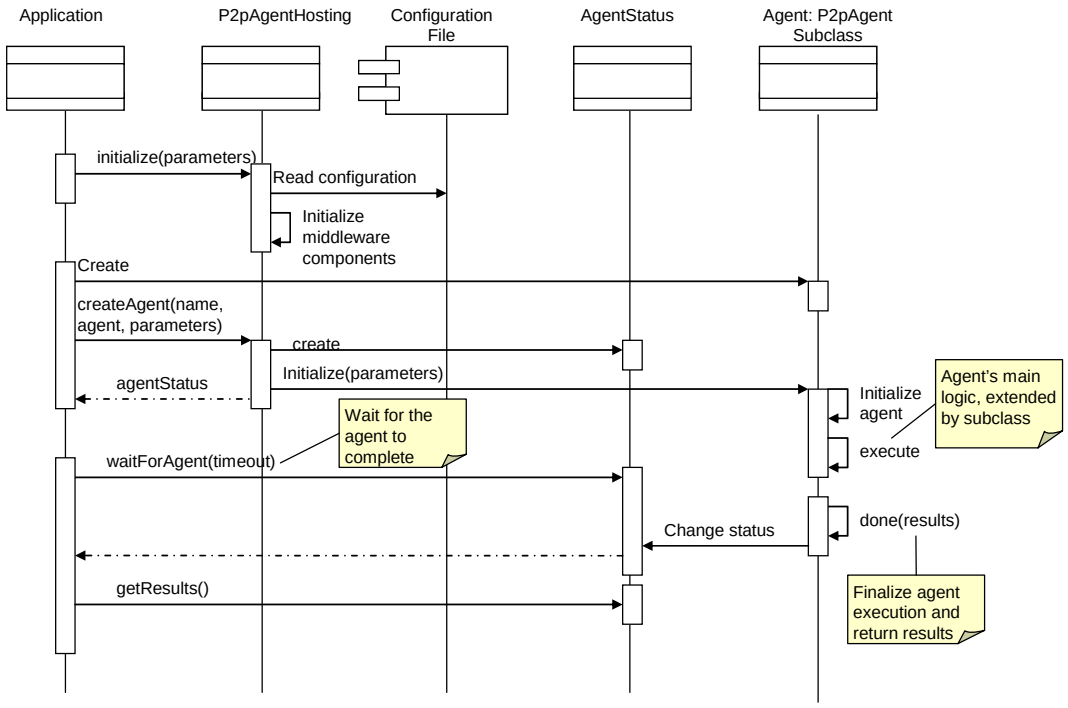

Figure 5-6 Programming model

The middleware also offer two communication models for agents: a direct agent-to-agent model and a group communication mechanism

In the direct communication model, an agent is able to send a message to a specific destination agent, in the same P2pAgentHosting environment on in a remote machine, given it has an AgentReference to that agent, which encodes all the information needed to reach the destination. These references are provided for local agents by the P2pAgentHosting given the name of the agent. The middleware does not provide any service to look for references of

remote agents, but applications are also free to provide its own directory service to store and look for such references.

The messages are delivered to the destination agent by the middleware, and it is responsible to handle in the handleMessage method. To allow complex interactions among agents, the API provide methods to reply to the originator of a message or to forward it to a third agent, which in turn could reply to the originator or the intermediate agent. This interaction model is shown in Figure 5-7.

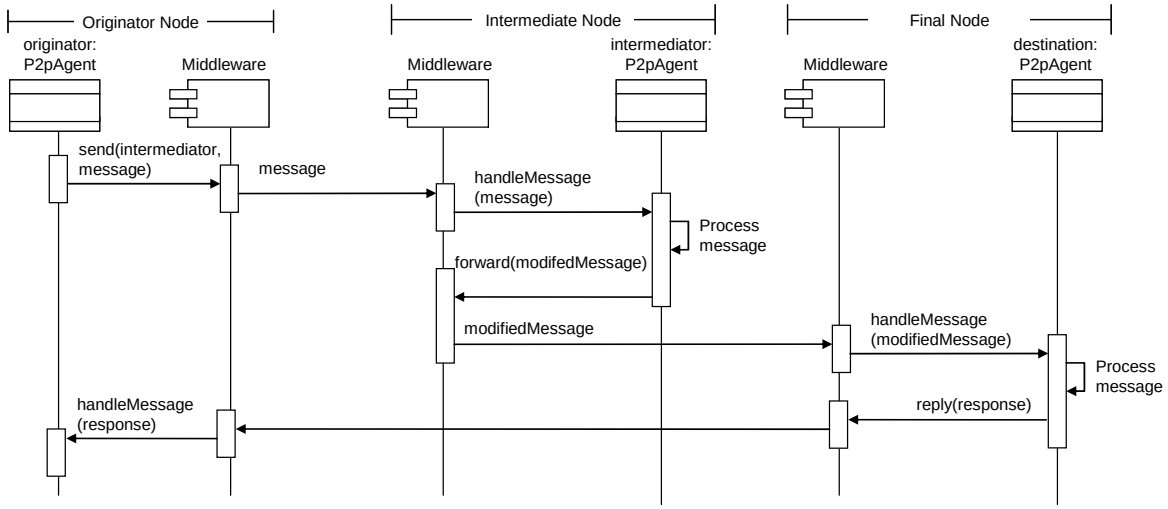

Figure 5-7 - Interaction Model

Finally, the middleware provides a group communication mechanism. Agents can join one or more groups, which are identified by a globally unique name. Once the agent has joined the group, it will receive messages sent by other agents to this group, using the handleGroupcast method. All agents in a group should receive messages, however, the middleware currently does not provide guaranties of delivery, nor does it allow detecting missing messages. Messages can be sent to a group by any agent, even if it has not joined the group, using the groupcast method, which takes as parameters the name of the group and the message. This model is presented in Figure 5-8.

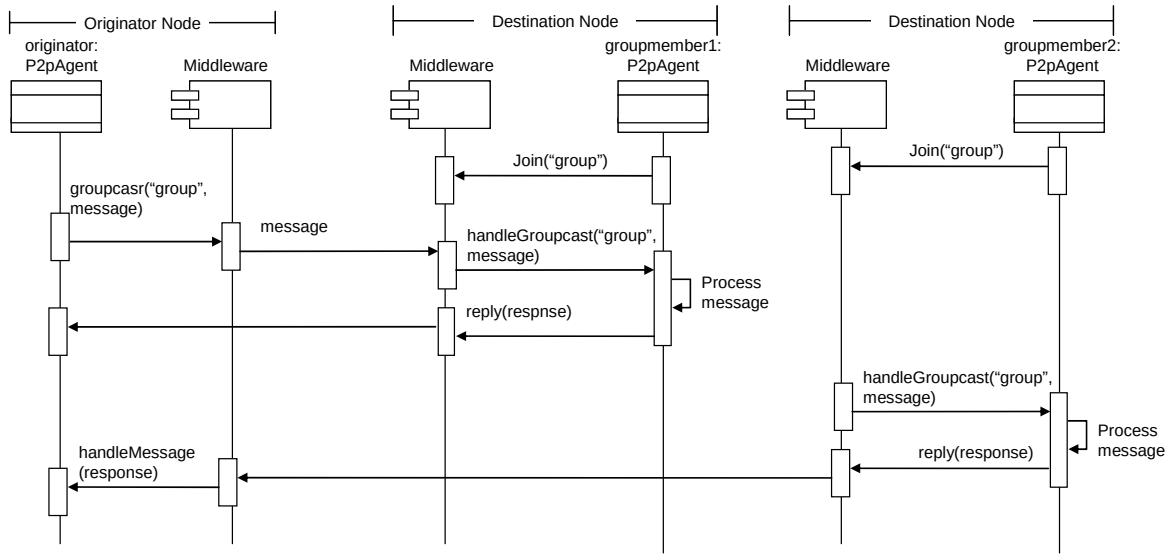

Figure 5-8 - Groupcast model

Interestingly, messages received in a groupcast carry the agent reference of the originator and can be also replied and forwarded, to allow complex interaction patterns.

5.2.3 Resource Management integration

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.

The implementation of this simple resource manager is based on standard Linux interfaces that give access to counters on which the kernel keeps track of the resource utilization. More specifically, the special file /proc/stat is read at initialization time (time t0) when the CPU utilization is requested (time tn) and the following formula is applied:

$$\text{Utilization}(tn) = \frac{\text{total cpu used } (tn) - \text{total cpu used } (t0)}{\text{total cpu time}(tn) - \text{total cpu time } (t0)}$$

Where "total cpu used" is the sum of the time spend running both system and user level processes and the "total cpu time" is the time elapsed from the system start up. This is a fairly common approach to report cpu utilization in Linux environments. Similar approaches could be applied to other resources, as memory and disk space.

This simple interface offers some important advantages for the overall architecture:

- The resource agent has the freedom to select the frequency of monitoring and can decide to use instantaneous values or an average over a period, without requiring changes in the resource manager
- Clearly separates the pricing from the resource monitoring, allowing different pricing models to be used, even simultaneously
- Does not depend on an exclusive control of resources, as other workload not controlled by the middleware can still be processed.
- The evaluation of the allocation economic based strategies can be done without the interference of other, potentially conflicting, allocation policies used by sophisticated resource managers.

Currently the main drawback of this approach is the impossibility to warrantee the resources allocated to a process, as the resource manager cannot enforce any utilization policy and therefore the allocation follows a "best effort" model.

5.3 Deployment and test framework

5.3.1 Deployment framework

When dealing with distrusted systems, deployment is a key step to success. Whereas in a single machine a deployment translated to the single installation of several components in one machine, in a distributed systems the deployment implies concurrent installation/setup in several machines, simultaneous verification and eventually test on the communication channels. The deployment must provide also the right tools to remotely launch operations on the distributed systems.

In the scope on year 2 development on GMM in CATNETS, we have built up several linux scripts allowing for the deployment of the GMM in a generic subnet of Linux machines. These scripts allow for the following functionalities:

- deployment of GMM infrastructure: middleware executable jar, required libs, configuration and launching scripts
- deployment of Web Services in remote nodes
- remote configuration/launching of Basic Service/Resource nodes
- execution/monitoring of Client requests
- Traces and metrics re-collection

The scripts use basic unix shell scripting language and vxargs (http://freshmeat.net/projects/vxargs/) to provide parallel execution of this scripts in remote nodes, as well as to capture the resulting output files

All the resources needed to perform these operations are bundled in the GMM package consisting in:

- middleware executable, supporting libraries and configuration files
- apache tomcat Web Server
- apache axis for Wed Services and supporting libraries
- Scripts to deploy components, run experiments and collect results
- Classes and data for J48 Data Mining Services
- List of nodes where the experiments will be run

The deployment operation is depicted in Figure 5-9:

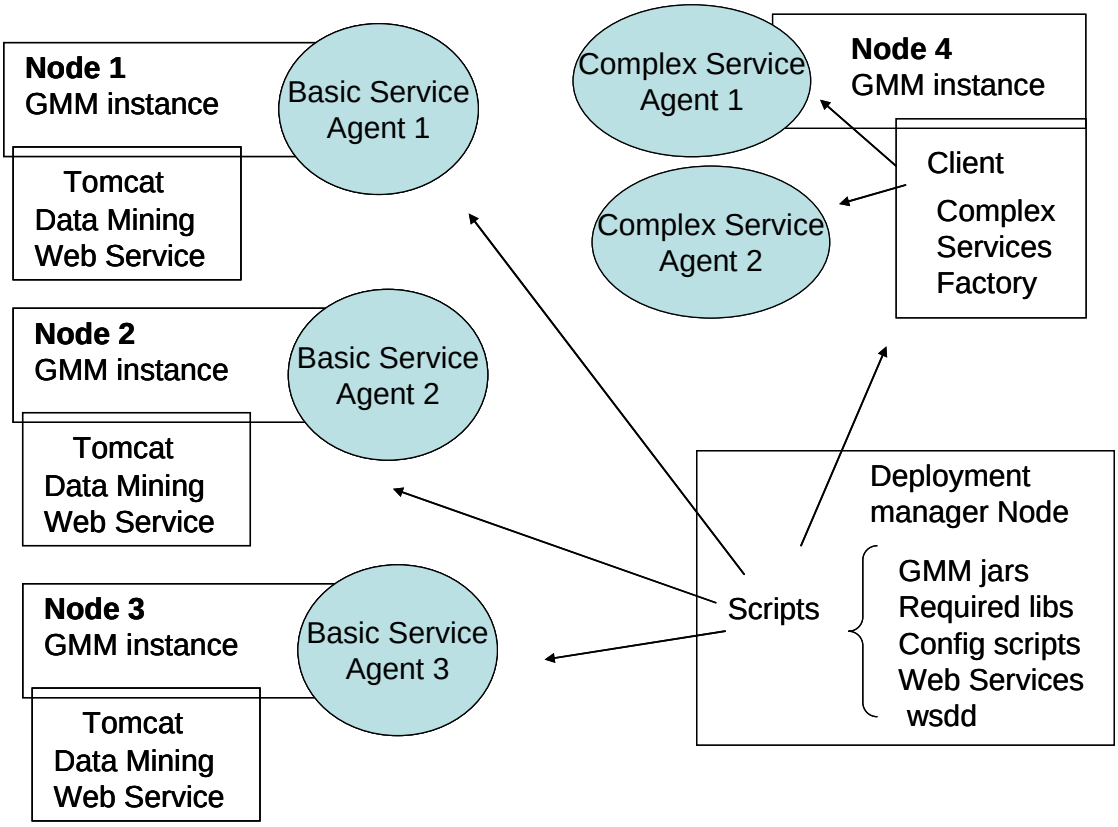

Figure 5-9 Deployment process

To deploy the middleware and execute an experiment on a list of machines, it is necessary to follow these steps:

1. Define the list of nodes that will participate on the test
2. Deploy the middleware and required data in all nodes in nodeList.txt
3. Launch the middleware on the nodes
4. Check the evolution of the middleware using the files generated by vxargs with the console and error outputs from each node (these files are continuously updated by vxargs)
5. When the experiment finishes, collect logs and metrics

Listing for the UNIX/Linux scripts is found in annex A1.

A typical sequence of events during the middleware operation is presented in Figure 5-10.

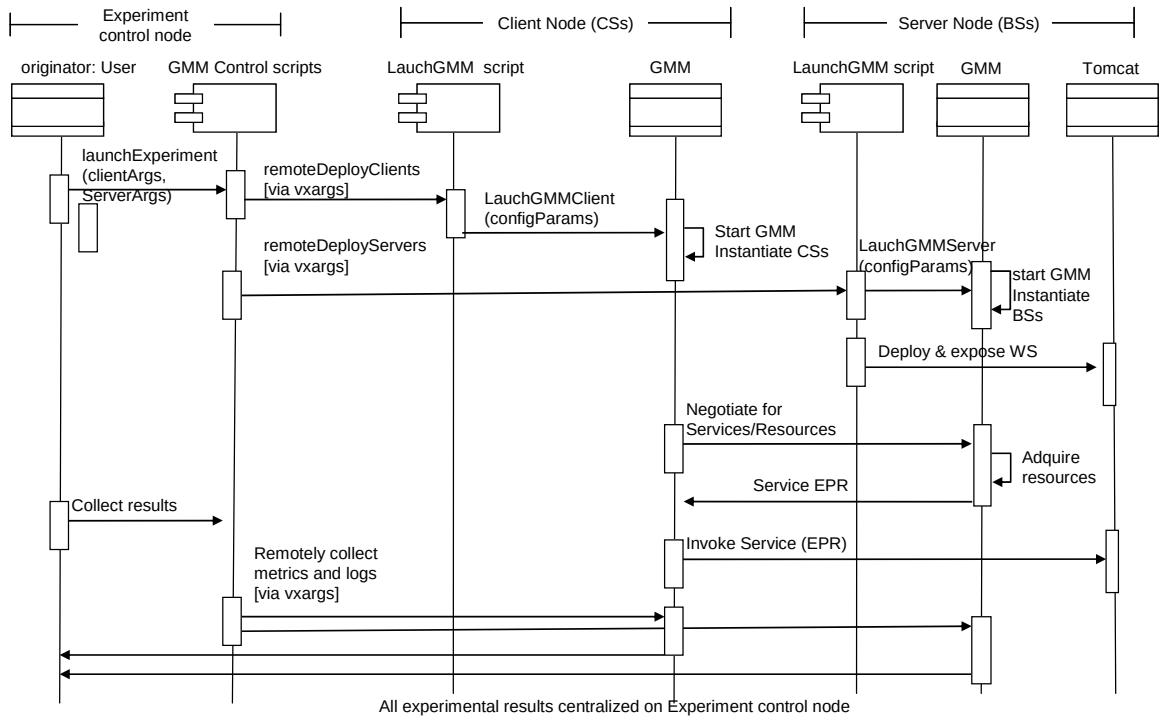

Figure 5-10- GMM Use Case

From this diagram we can see that is fairly straightforward for a user to perform an experiment by using the provided scripts. Just by launching the launchExperiment script, giving the convenient configuration options (list of Clients and servers, and GMM configuration files), and all the operations are triggered automatically. Experiment evolution can be monitored in the Experiment Control Node at any time using vxargs. Once experiment is over, remote results collection is also straightforward by launching the appropriate script from the Experiment Control Node.

The scripts are configurable to allow for the deployment of different Web Services on the server nodes, and different middleware versions can be deployed just by providing the right middleware jar. The scripts for deployment of GMM and experimental control are integrated into the GMM development releases.

5.3.2 Test framework

Using JMeter (see section 5.1.3) we have developed a test framework for the middleware. This provides the possibility of executing controlled tests on the middleware taking advantage of the JmMeter features, as repeated tests, concurrent tests, among othersThe integration with JMeter is done by means of a extension of the JavaSamplerClient class. This class is a general test controller with offers generic functionalities needed by any test to be performed in the middleware as:

- Load a configuration file to describe test
- Initialize the middleware locally on the test driver node
- Execute an action on all the nodes described in the configuration file
- Select randomly a node from the configuration file to execute a test action
- Invoke the TestAccessPoint web service (described below) on the target node to execute an action

A test is then defined as a series of invocations to the MiddlewareTestSampler, passing on each invocation the name of the action to be performed and action specific parameters. These parameters are passed using a facility provided by JMeters, which allows the specification of test parameters for each action in a test, as is shown in Figure 5-11.

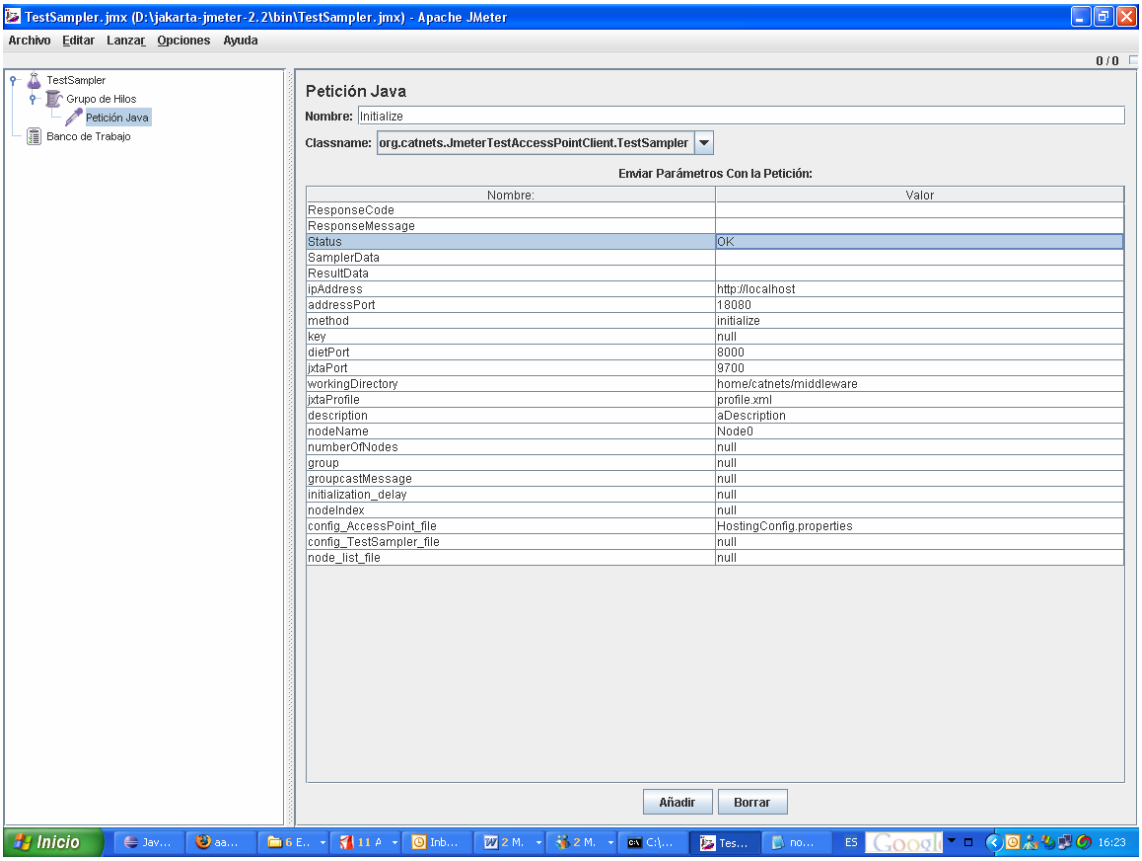

Figure 5-11 Parameter Definition for a Test Action

The MiddlewareTestSampler requires at initialization a list of the test nodes specifying:

- address and port of the Test Access Point web service
- Configuration parameters for each node

An example of this configuration file is given in Listing 5-1.

```
nodes.nodelist node0, node1

node.node0.name Local Node
node.node0.config_AccessPoint_file HostingConfig.properties
node.node0.ipAddress http://localhost
node.node0.port 18080
node.node0.description aDescription
node.node0.workingDirectory D:\\pruebasMiddleware
node.node0.jxtaProfile profile.xml
node.node0.jxtaPort 9700
node.node0.dietPort 8608

node.node1.name Remote Node
node.node1.config_AccessPoint_file HostingConfig.properties
node.node1.ipAddress http://remotehost
node.node1.port 18080
node.node1.description aDescription
node.node1.workingDirectory /home/ichao/pruebasMiddleware/
node.node1.jxtaProfile profile.xml
node.node1.jxtaPort 9701
node.node1.dietPort 8602
```

Listing 5-1 Test Configuration File

The TestAccessPoint web service is responsible of the initialization of the node and the invocation of test actions. For each test action, it receives from the MiddlewareTestSampler as parameters the name of the action to be performed and the action specific parameters, as a list of key, value pairs. It executes the desired action, returns the results and reports any error information. This approach allows the extension of the test actions just by providing the required test agents, without any modification of the TestAccessPoint.6

Middleware Initialization

Before the test can be started, all the participant nodes must be initialized. This is done by invoking the initialize action on all the nodes described in the configuration file. Figure 5-12 . Initialization Processdescribes the steps of this initialization process.

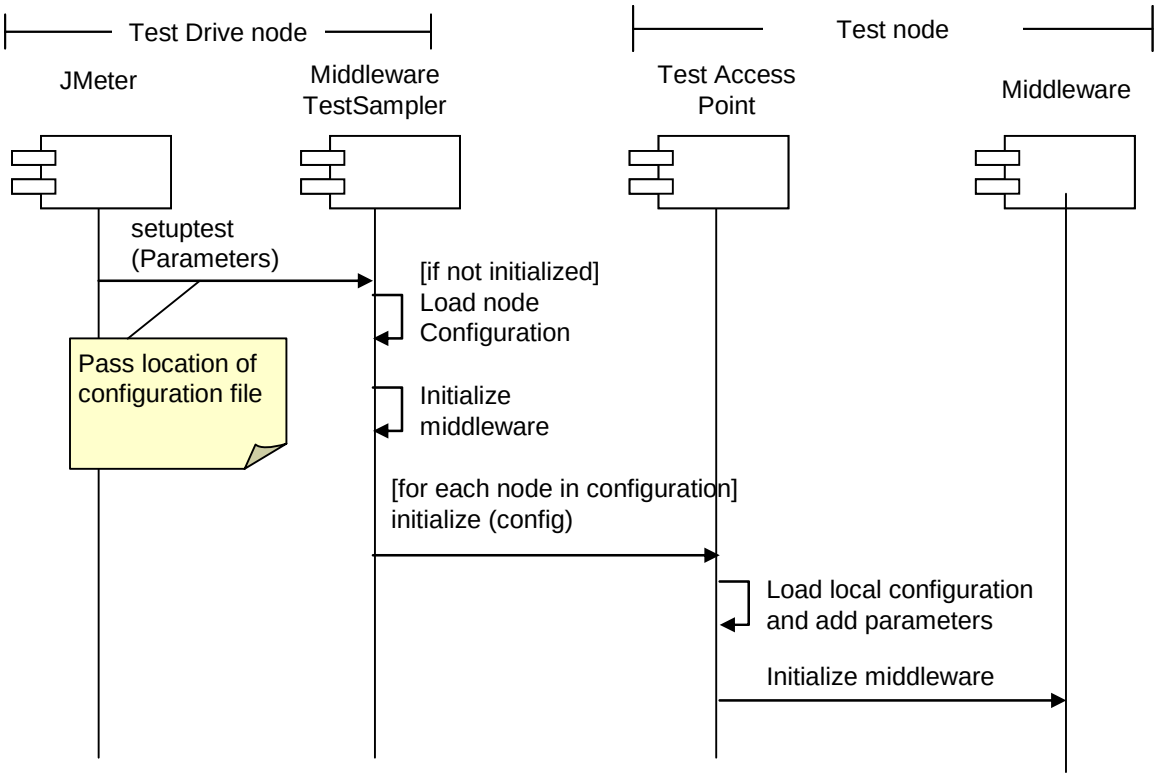

Figure 5-12 . Initialization Process

Test Action Execution

For each request to execute a test action the TestAccessPoint web service creates a request specific agent (given as part of it configuration) and passes the parameters it received, waits for a response from the request agent and returns a response to the MiddlewareTestSampler as a map of key-value pairs. Figure 5-13 - Performing a Test Action Figure 5-13 shows this process.

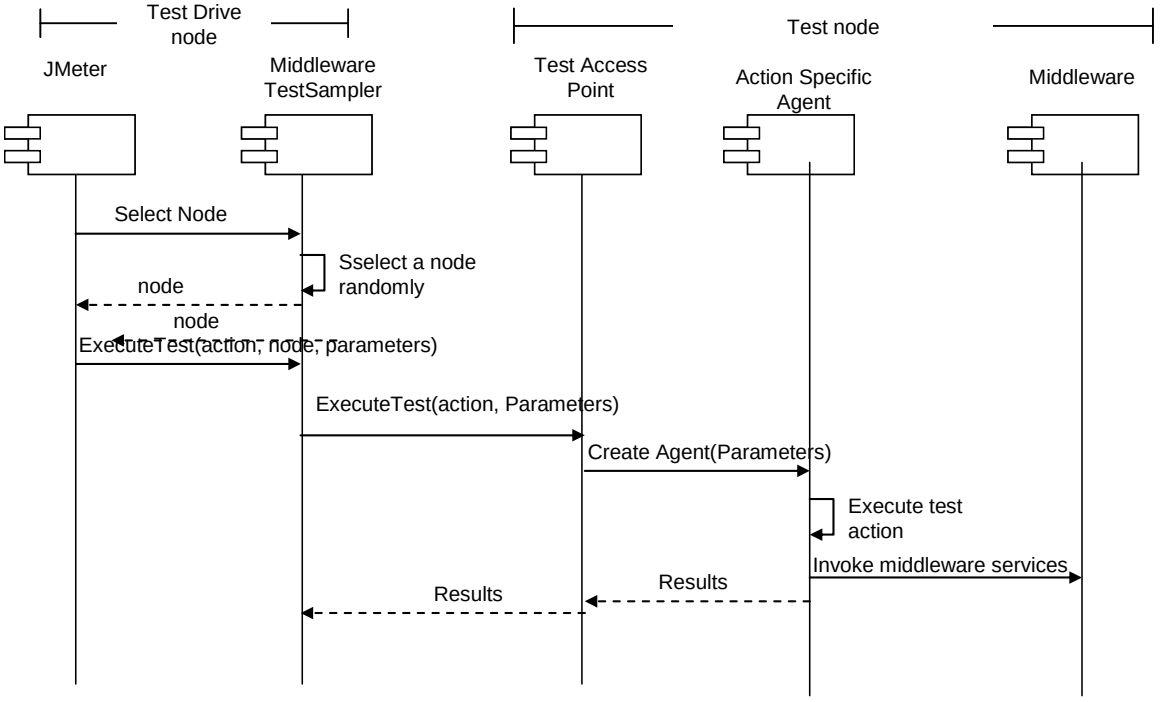

Figure 5-13 - Performing a Test Action

6 Prototype evaluation

6.1 Introduction

6.1.1 Objectives

We have performed several experiments in a cluster with the current version of the middleware, allowing for a preliminary performance assessment of the GMM.

The goal of the experiments is to evaluate the autonomic behaviour of the GMM in terms of self-organisation, given by decentralized resource discovery, and in terms of self-optimisation, given by adaptation to load and capacity of the resources.

6.1.2 Economic Algorithms

For these experiments we have used for the economic agents an implementation of the ZIP (Zero Intelligence Plus) agents (PRIEST94) which use a gradient algorithm to set the price for resources. Clients initiate negotiations with a price lower than the available budget. If they are not able to buy at that price, they will increase their bids until either they win or reach the budget limit.

Services will start selling the resources at a price which is solely influenced by the node's utilization, following the pricing model presented in (YEOP04). As Services get involved in negotiations, the price will also be influenced by demand. If a Service agent is selling its resources, it will increase the price, when resources are sold to identify the price the market is willing to pay. When it no longer sells, it will lower the price until it becomes competitive again or it reaches a minimum price defined by the current utilization of the resource. This protocol is illustrated in Figure 6-1.

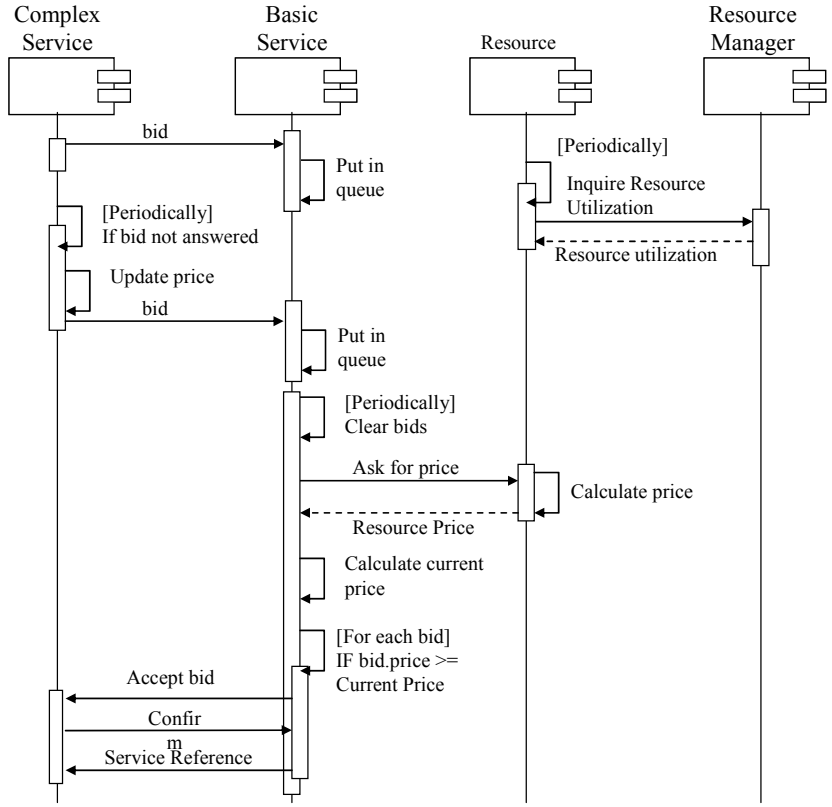

Figure 6-1 Bidding protocol

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 "tests" the market by raising the price, looking for higher profits.

To determine how much the price must be adjusted, the basic service first calculates a target price as follows:

```
IF CurrentPrice > MaxBid THEN
        TargetPrice = CurrentPrice *(1-PRICE_STEP)
ELSE
        TargetPrice = MaxBid*(1+PRICE_STEP)
ENDIF
```

Where PRICE_STEP is a parameter that defines the price change ratio. For the experiments being reported in this document, this ratio is set to 0.05 of the current price.

Then, the basic service uses a gradient algorithm to define the speed at which it will move to this new target price. This prevents the agent over reacting to sudden price rises or falls. The change in price is calculated as:

```
PriceChange = LEARNING_GAMMA*CurrentPrice +
        LEARNING_GAMMA)*LEARNING_BETA*
        (CurrentPrice-TargetPrice)
```

Where LEARNING_BETA and LEARNING_GAMMA are parameters that control the speed at which the algorithm incorporates the new information about changes in the target price. For the experiments being reported, these parameters were set to 0.7 and 0.05 respectively.

Finally, the price is updated according to the price change and the change direction (-1 if the price must be lowered and +1 if it must be raised):

```
PriceUpdate = CurrentPrice+ChangeDirection*PriceChange
```

6.1.3 Data Mining Services scenario

The Web Services deployed on the nodes of the cluster correspond to a simplified version of the Data Mining scenario described in detail section 3.2. We have deployed J48 classifier Web Services. The J48 algorithm applies to a data set specified by the user. The data set must be in the ARFF, and we also expose it in application servers.

6.2 Experimental setup

In order to test the performance of the market based resource allocation mechanism, we setup controlled experiments deploying several instances of the middleware in a Linux server farm.

Each node has 2 CPU Intel PIII 1 GHz and 512 MB of memory. The nodes in the farm are connected by an internal Ethernet network at 100Mps.

We deploy GMM and the Web Services on three nodes (named arvei-10, arvei-11 and arvei-12). On each node we also deployed a Web Service which performs a CPU intensive calculation on the machines, increasing load. These Web Services are exposed in a Tomcat server. Access to execute these Web Services on the Resources is what will be negotiated by the Service with the Clients. Figure 6-2 shows a schematic view of this deployment.

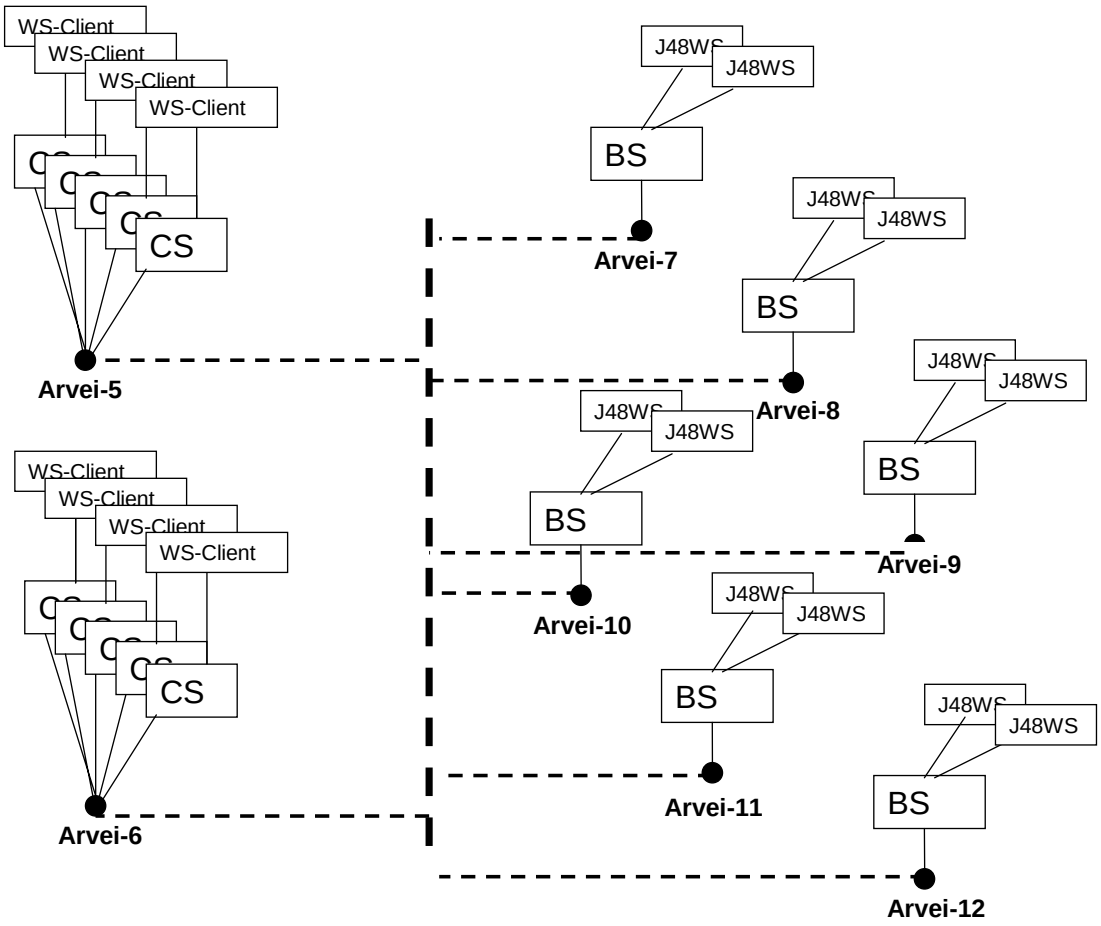

Figure 6-2 - Experimental setup

Realistic but still controllable experimental conditions can be achieved by using load generators. We have used lookbusy, a simple tool for generating synthetic load on a Linux system. It can generate fixed, predictable loads on CPUs. When generating CPU load in particular, lookbusy will attempt to keep the CPU(s) at the chosen utilization level, adjusting its own consumption up or down to compensate for other loads on the system.

The experiments consisted in launching 2 Clients concurrently from 2 other nodes which were not running the Web Services. We generate a baseline load on all three nodes of 25% of CPU usage to simulate some background activity. The load is generating using Each Client performs 50 requests, in intervals of 10 seconds. Whenever a Client wins a bid with a Service, it invokes the Web Service in the selected node. The complete experiment runs for about 10 minutes. To better test autonomic load balancing, we artificially stressed one of the nodes (arvei-10) up to 95-100% of CPU usage for a short time during the experiment.

6.3 Performance Results

Figure 6-3 shows how the prices of services are determined by the utilization of the resource, and this in turns determines the number of service allocations (after successful negotiations).

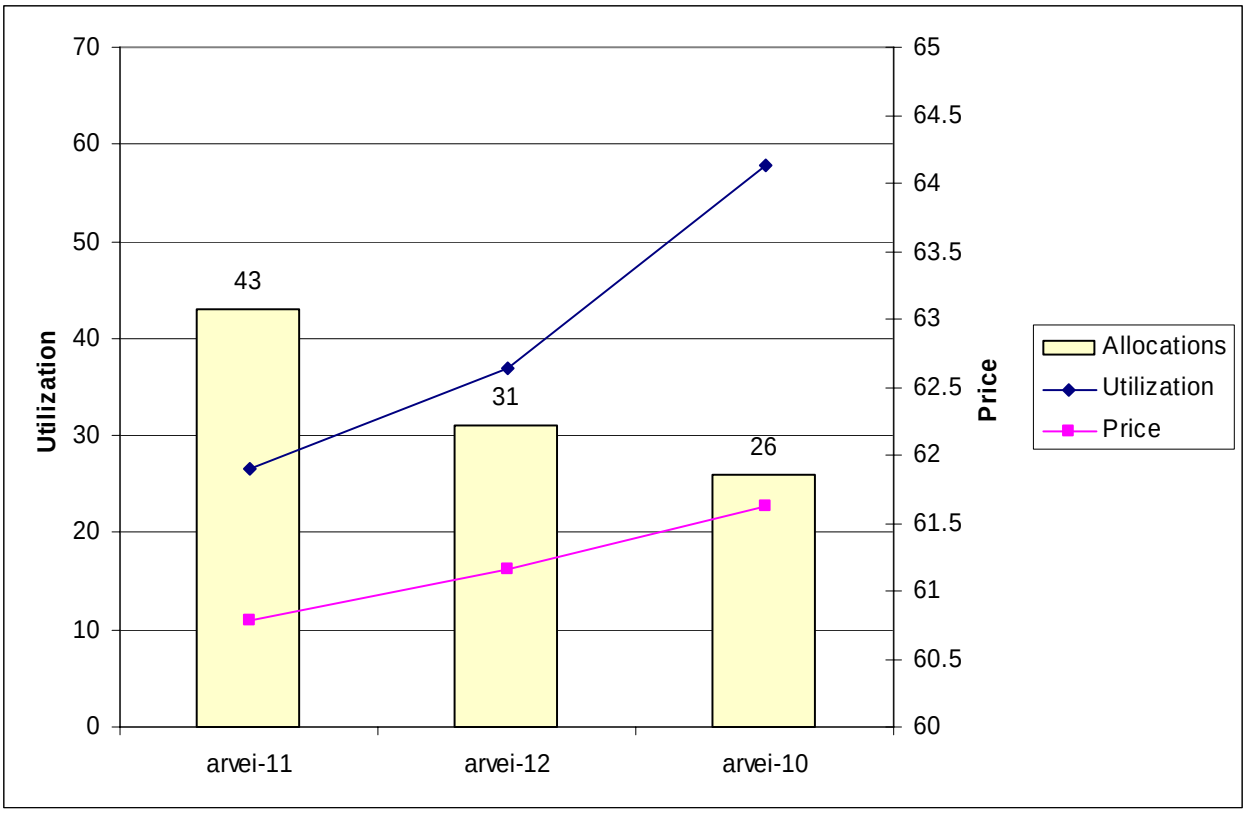

Figure 6-3 - Price – Utilization relationship

In Figure 6-4 we can see time series for the evolution of utilization and prices on the server nodes. We can clearly identify how arvei-10 has a period of high load, which corresponds to the artificially introduced stress.

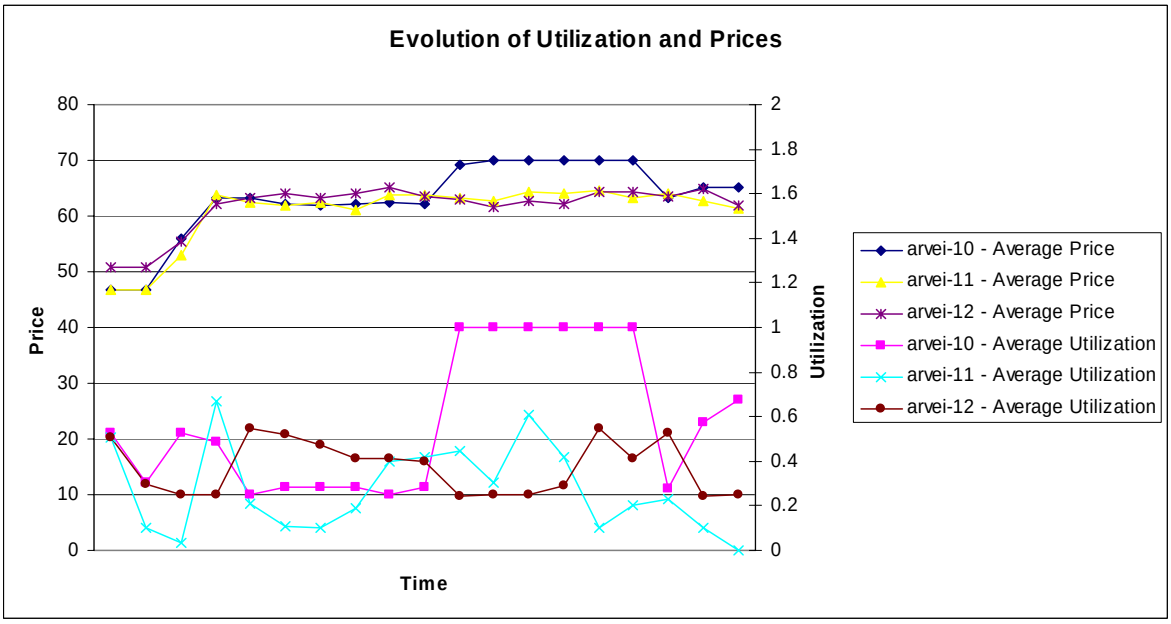

Figure 6-4 Evolution of utilization and prices

We can see the resulting increase in market price for this node. In contrast, in the other nodes, the load and prices vary smoothly as the load varies due to the execution of the services.

Figure 6-5 presents the number of allocations made for each service. It shows how the decentralized economic negotiation leads to a load balancing of the system: nodes with lower utilization made more allocations, thus showing a degree of self-organization. Arvei-10 reacts to the sudden load stress by decreasing its allocation rate for the period with high load.

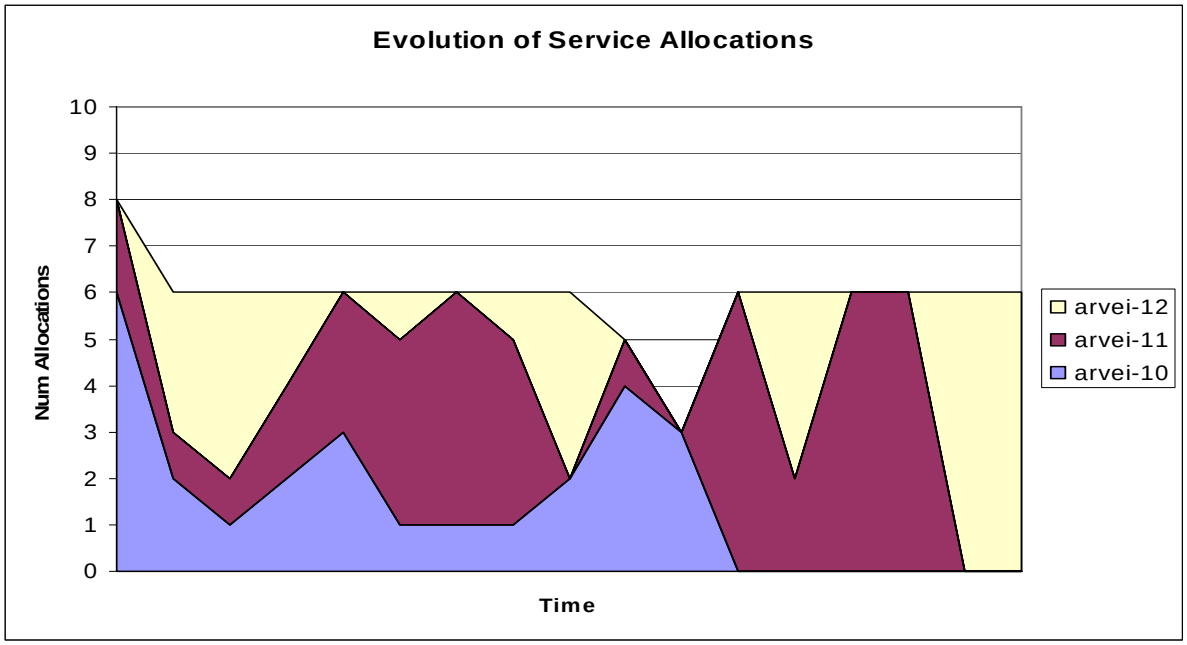

Figure 6-5: Evolution of service allocations (number of allocations versus time)

Figure 6-6 shows the service execution time and negotiation time (time employed by the Catallactic middleware to negotiate for the service execution). It can be seen that the execution time remains approximately constant over the runs, due to the achieved load balancing on the nodes.

Figure 6-6: Evolution of the negotiation time and execution time during an experiment.

6.4 Evaluation of an uncontrolled environment

As part of the evaluation of the prototype, we experimented in an environment shared with other users, where the total workload of nodes was not controlled in the experiments and was, in general, fairly high (well above 80%). The test was carried out using 9 Linux servers, each with 2 Xeon processors and 2GB of memory The Basic Services where deployed on seven servers, and Complex services where launch on two servers. Figure 6-7 shows this setup.

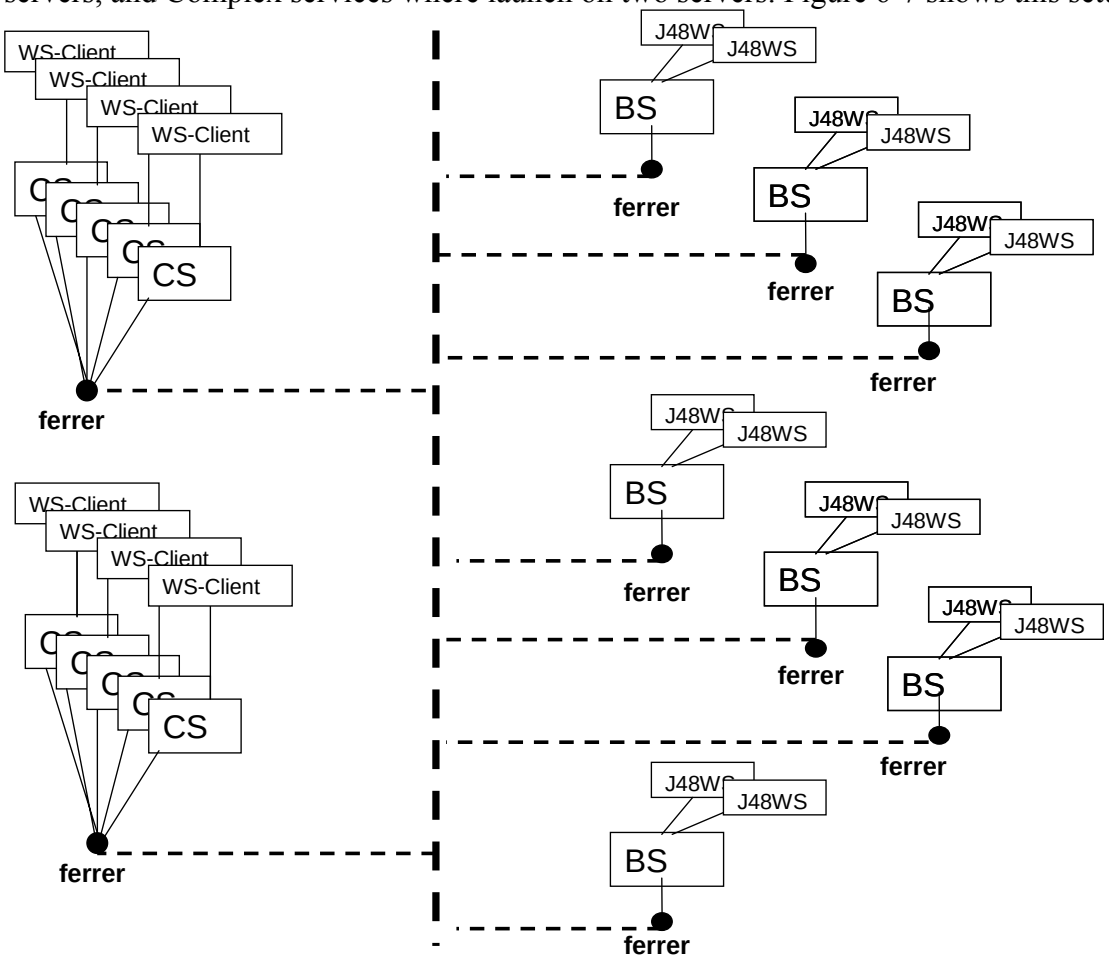

Figure 6-7 - Setup of experiment in uncontrolled environment

Experiments consisted in launching 9 requests from each client at an interval of 5 seconds. Due to the high utilization of a server, service allocations where concentrated in the two less loaded servers, as can be seen in Figure 6-8.

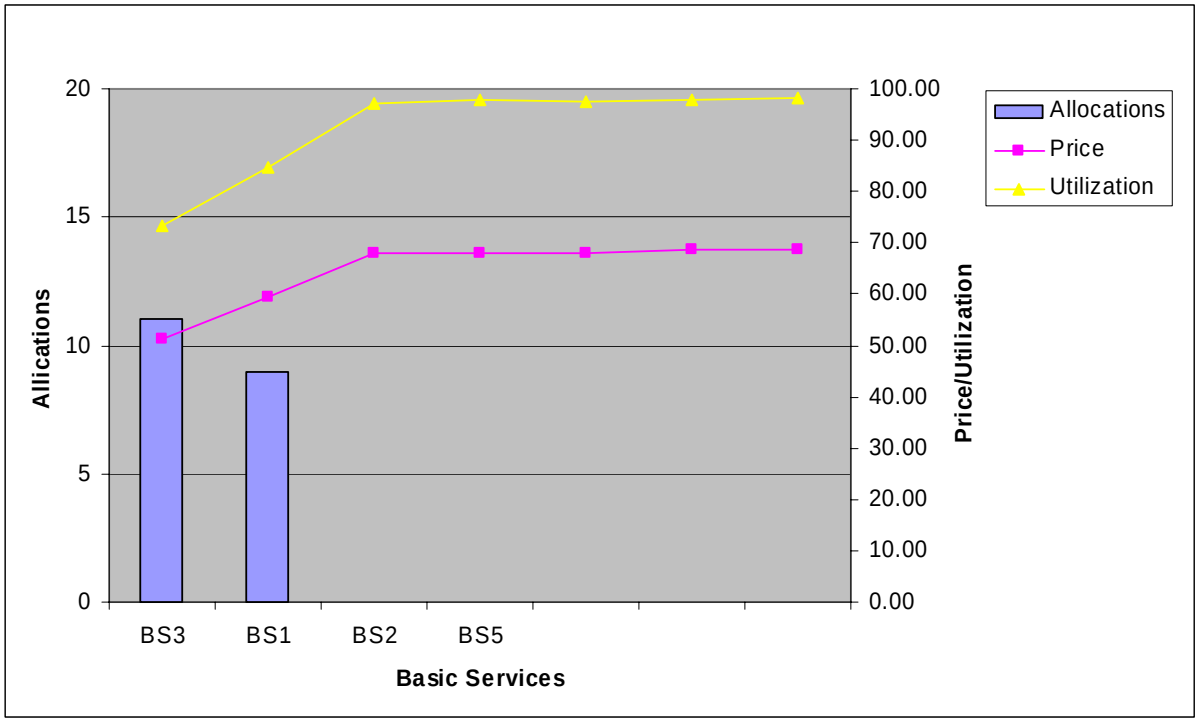

Figure 6-8 - Allocation in uncontrolled environment

However, the service execution time was maintained very stable along the execution of the experiment, as is shown in Figure 6-9.

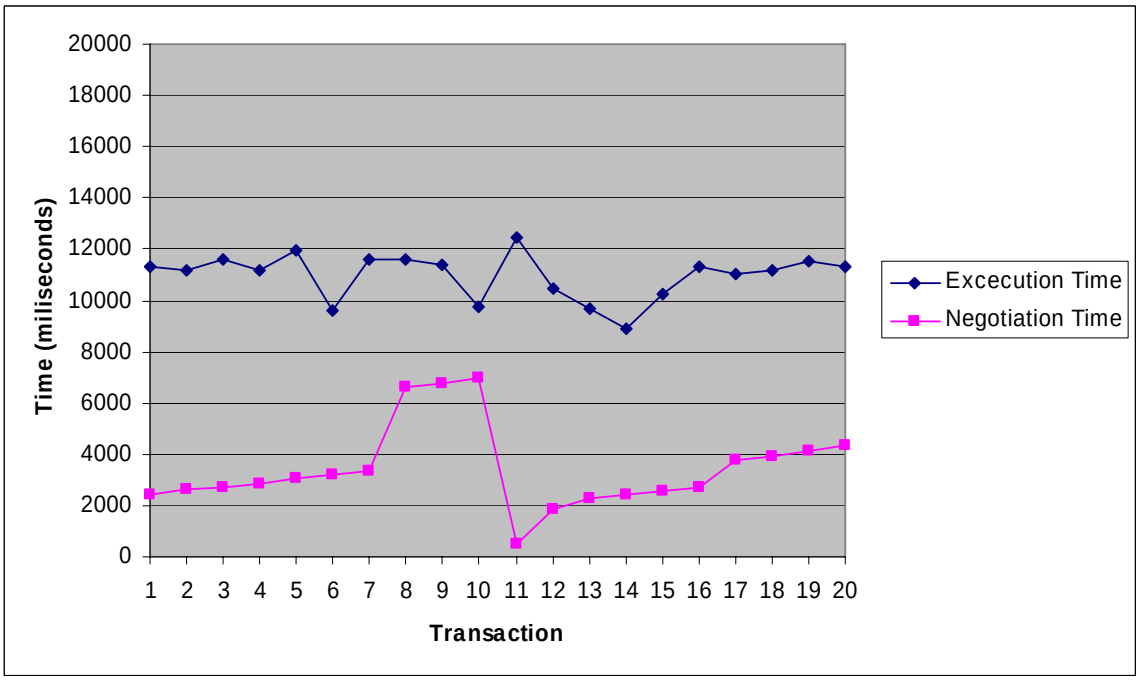

Figure 6-9 - Service negotiation and execution time in uncontrolled environment

7 Future Work

7.1 Triana Integration

Triana is a graphical environment that allows creating and assembling a workflow process from a set of building-blocks/units. Triana was implemented and used in the European FP5-funded "GridLab" project, and has been utilized by a number of application scientists – ranging primarily from Astrophysicists involved in analysing Gravitational wave data from laser interferometers, to bio-scientists interested in investigating biodiversity data. The building blocks used in Triana could represent different services, complex tasks or operations that the above prototype applications require.

- Triana uses a wide variety of data: numerical data, audio data, images, or text files.
- Triana allows displaying data either in text-editor data window or in a graph-display window.
- Triana allows inserting display units to monitor intermediate results.

The features of Triana will be used in order to support case 1 (see section 3.1) and further investigation will be done to support case 2 (see section 3.2) presented above, in which a user could visually describe the workflow. Subsequenrly, all the components of the prototype could be mapped to units that describe the services.

Both Cat-COVITE and Data Mining prototype applications are expected to be integrated into the Triana workflow engine in year 3. In section 7.1.1 details of work that has so far been undertaken using Triana is provided, and the first deployment scenarios are shown.

7.1.1 Deployment Scenarios

1. Cat-COVITE Prototype

Figure 7.1 shows a snapshot of the Cat-COVITE prototype – showing the interaction between the Master Grid Service (MGS) module at the application level, and the GMM (see Figure 2.3) via the CAP. This interaction refers to the use case described in section 3.1.

Figure 7.1 details:

1 – these methods are part of CAP and their functionalities are to deal with the agreement template requests and agreement offers received from the MGS;

2 – this method is part of the MGS and is responsible for job execution, if an EPR is received.

3 – these modules are part of MGS.

The ApplicationInstance, UserLocation and Job are attributes at the application level and are part of the MGS module of the prototype. The MGS is responsible for interaction with the CAP to finding services/resources on the Catallactic markets.

The prototype operates as follows: the Cat-COVITE contains the MGS module, which interacts with the CAP Web Service, and this returns a static endpoint reference of the service needed by the prototype to fulfil the job. The CAP in Figure 7.1 is not linking to any Catallactic GMM, therefore currently a static string is returned instead. This demonstrates the

connection and interaction between the prototype and the CAP, as the main access point to services / resources markets.

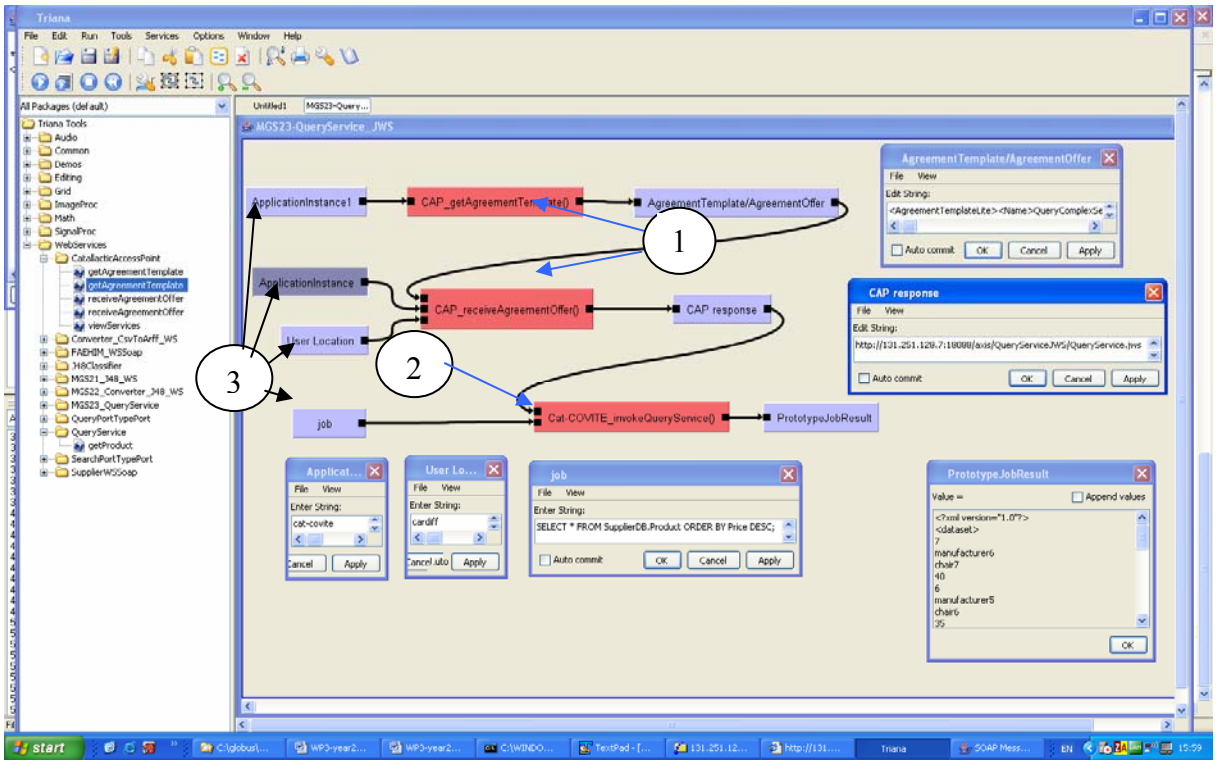

Figure 7-1 – CAT-COVITE -- interactions between MGS, CAP and Catallactic middleware (GMM)

The interaction between the components shown in Figure 7.1 is as follows:

- CAP_getAgreementTemplate() and CAP_receiveAgreementOffer() are part of the CAP Web Service. They are required to handle the interaction between the Application Prototype Instance and the Catallactic GMM by a complex service request.

The listings below show how the data is passed between the services -- encoded in the SOAP protocol.

```
... <wsa:To soapenv:mustUnderstand="0">
      http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
    </wsa:To> ...
```

Listing 7.1 – CAP invocation

```
... <serviceName xsi:type="xsd:string">
      cat-covite
    </serviceName> ...
```

Listing 7.2 – Service Instance invocation

- The WS-Agreement protocol is used to exchange the Templates and Offers between the components (MGS and CAP).

```
... <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
&lt;AgreementTemplateLite&gt;&lt;Name&gt;QueryComplexService&lt;/Name&gt;&lt;Context&gt;&
lt;AgreementInitiator&gt;&lt;/AgreementInitiator&gt;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;
TerminationTime&gt;&lt;/TerminationTime&gt;&lt;/Context&gt;&lt;Terms&gt;&lt;BasicServiceType
&gt;QueryService&lt;/BasicServiceType&gt;&lt;NumberOfBasicServiceNodes&gt;1 to 10&lt;!--
```

between 1 to 10 --
></NumberOfBasicServiceNodes><BasicServiceConstraints><DBType>Architectural/Engineering/Construction</DBType><ResponseTimePerRequest>10</ResponseTimePerRequest></BasicServiceConstraints><PayForService></PayForService></Terms></AgreementTemplateLite>
</ns1:getAgreementTemplateReturn> ...

Listing 7.3 – Application and CAP interaction – agreement template uploaded by the application instance

... <agreementOffer xsi:type="xsd:string">
<AgreementOfferLite><Name>QueryComplexService</Name><Context><AgreementInitiator>cardiff</AgreementInitiator><StartingTime></StartingTime><TerminationTime></TerminationTime></Context><Terms><BasicServiceType>QueryService</BasicServiceType><NumberOfBasicServiceNodes>1 to 10<!--
between 1 to 10 --
></NumberOfBasicServiceNodes><BasicServiceConstraints><DBType>Architectural/Engineering/Construction</DBType><ResponseTimePerRequest>10</ResponseTimePerRequest></BasicServiceConstraints><PayForService>100</PayForService></Terms></AgreementOfferLite>
</agreementOffer> ...

Listing 7.4 – Application and CAP interaction – agreement offer sent by the application instance

- The value returned from the CAP is either an end point reference (EPR) or nothing.

... <ns1:receiveAgreementOfferReturn xsi:type="xsd:string">
http://131.251.128.7:18088/axis/QueryServiceJWS/QueryService.jws
</ns1:receiveAgreementOfferReturn> ...

Listing 7.5 – Application and CAP interaction – the decision is returned from GMM – EPR in this case

- If an EPR is returned, the prototype instance uses it to complete the application task.

An application job is encoded in the SOAP message and sent to the EPR and the result is return to the user.

... <job xsi:type="xsd:string">
*SELECT * FROM SupplierDB.Product ORDER BY Price DESC;*
</job> ...

Listing 7.6 – Application job submission

... <ns1:invokeQueryServiceReturn xsi:type="xsd:string">
<?xml version="1.0"?>
<dataset>
7
manufacturer6
chair7
40
6
manufacturer5
chair6
35
5
manufacturer4
chair5
30
4

```
manufacturer4
chair4
25
3
manufacturer3
chair3
20
2
manufacturer2
chair2
15
1
manufacturer1
chair1
10
&lt;/dataset&gt;
      </ns1:invokeQueryServiceReturn> ...
```

Listing 7.7 – Result encoded in the SOAP message

Full details of the communication between the main components of the prototype, shown in Figure 7.1, are detailed in the Annex C.1.

2. Cat-DataMining Prototype

A. Reference to the use case described in section 3.1

An example of an application that does not integrate markets and it is using the Triana workflow composition with J48 data mining service is presented in FAEHIM toolkit [ALI05].

Figure 7.2 shows a snapshot of the Cat-DataMining prototype – the interaction between the Master Grid Service (MGS) – module at the application level, and the GMM (see Figure 2.3) via the CAP. As in Figure 7.1, the same methods and modules references apply.

Full details of the SOAP messages exchanged are shown in the Annex C.2.1. Below are highlighted parts of these messages, which show the main exchanged information, such as: what prototype instance is triggered, the CAP Web Service URI, agreement template and offer exchanged, the service URI received, the job, and the result returned to the client.

```
... <wsa:To soapenv:mustUnderstand="0">
      http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
    </wsa:To> ...
```

Listing 7.8 – CAP invocation

```
...   <serviceName xsi:type="xsd:string">
        cat-J48DataMining_WS
      </serviceName> ...
```

Listing 7.9 – Service Instance invocation

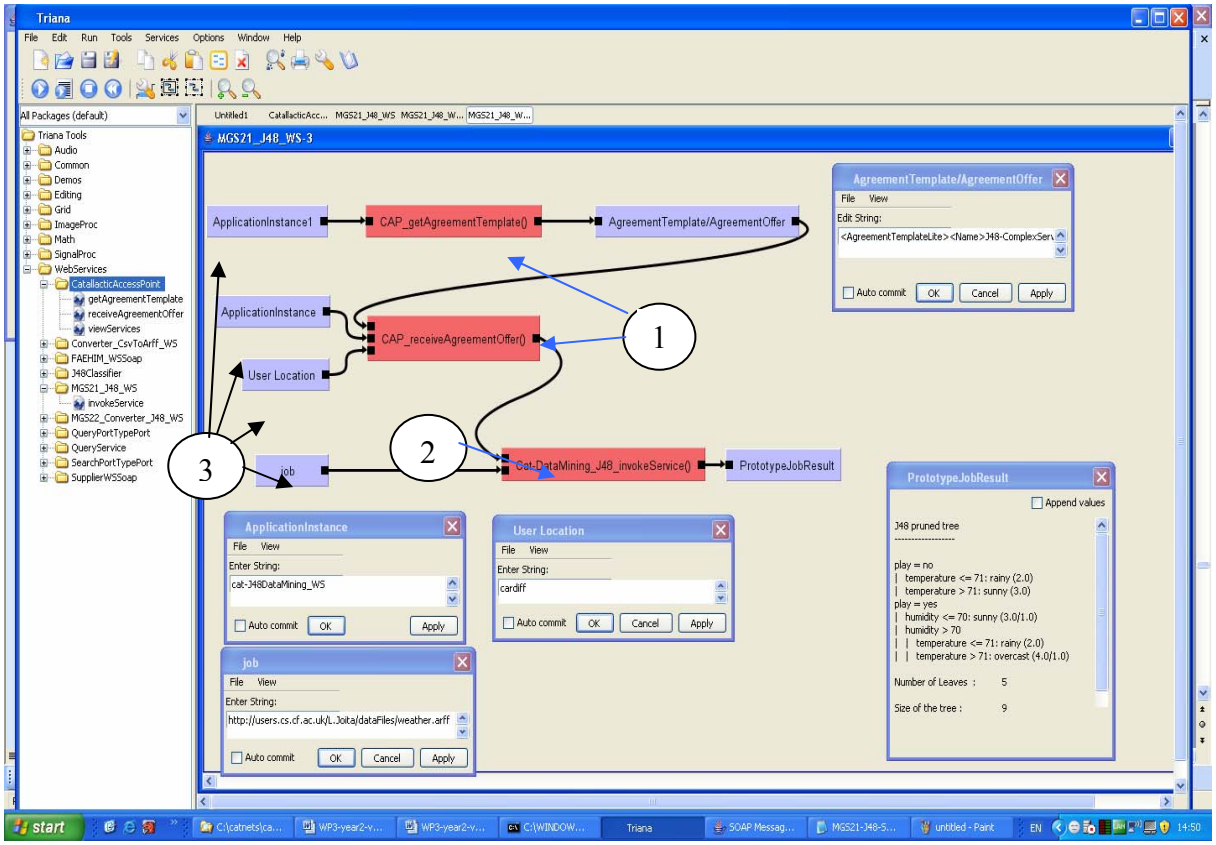

Figure 7.2 – Cat-DataMining -- interactions between MGS, CAP and Catallactic middleware (GMM)

... <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
<AgreementTemplateLite><Name>J48-ComplexService</Name><Context><AgreementInitiator></AgreementInitiator><StartingTime></StartingTime><TerminationTime></TerminationTime></Context><Terms><BasicServiceType>J48</BasicServiceType><NumberOfBasicServiceNodes><!-- between 1 to 10 --></NumberOfBasicServiceNodes><BasicServiceConstraints><ResponseTimePerRequest>10<!-- maximum milliseconds --></ResponseTimePerRequest></BasicServiceConstraints><PayForService></PayForService></Terms></AgreementTemplateLite>
</ns1:getAgreementTemplateReturn> ...

Listing 7.10 – Application and CAP interaction – agreement template uploaded by the application instance

... <agreementOffer xsi:type="xsd:string">
<AgreementOfferLite><Name>J48-ComplexService</Name><Context><AgreementInitiator>cardiff</AgreementInitiator><StartingTime></StartingTime><TerminationTime></TerminationTime></Context><Terms><BasicServiceType>J48</BasicServiceType><NumberOfBasicServiceNodes><!-- between 1 to 10 --></NumberOfBasicServiceNodes><BasicServiceConstraints><ResponseTimePerRequest>10<!-- maximum milliseconds --></ResponseTimePerRequest></BasicServiceConstraints><PayForService>120</PayForService></Terms></AgreementOfferLite>
</agreementOffer> ...

Listing 7.11 – Application and CAP interaction – agreement offer sent by the application instance

```
...     <ns1:invokeService soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
       <serviceURI xsi:type="xsd:string">
        http://131.251.128.7:18088/axis/services/J48Classifier
       </serviceURI>
       <job xsi:type="xsd:string">
        http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.arff
       </job>
     </ns1:invokeService> ...
```

Listing 7.12 – Invocation of EPR received from GMM and job submission

```
...      <ns1:invokeServiceReturn xsi:type="xsd:string">
        J48 pruned tree
------------------

play = no
|   temperature &lt;= 71: rainy (2.0)
|   temperature &gt; 71: sunny (3.0)
play = yes
|   humidity &lt;= 70: sunny (3.0/1.0)
|   humidity &gt; 70
|   |   temperature &lt;= 71: rainy (2.0)
|   |   temperature &gt; 71: overcast (4.0/1.0)

Number of Leaves  :    5

Size of the tree :       9

       </ns1:invokeServiceReturn> ...
```

Listing 7.13 – Result return to the user, encoded in the SOAP message

B. Reference to the use case described in the section 3.2 – Case 1.

Figure 7.3 shows a snapshot of the Cat-DataMining prototype – the interaction between the Master Grid Service (MGS) – module at the application level, and the GMM (see Figure 2.3) via the CAP. As in Figure 7.1, the same methods and modules references apply.

Figure 7.3 details the use case where a workflow of Web Services is constructed at the application level, and each service that composes it is requested individually to the CAP. The example in Figure 7.4 describes the following workflow process: Client (CSV data format) → ConverterCSV2Arff Service→ J48DataMining Service.

Snapshots of the SOAP messages exchanged are shown in the listings below. The full details of these messages are detailed in the Annex C.2.2. The highlighted parts of these messages show the main information exchanged, such as: what prototype instance is triggered, the CAP Web Service URIs (different CAPs are invoked for the Converter Web Service and the J48 data mining Web Service, but also the same CAP can be invoked if both services are discovered via the same market), agreement template and offer exchanged for individual Web Service requests, the service URIs received, the job, and the results returned to the client.

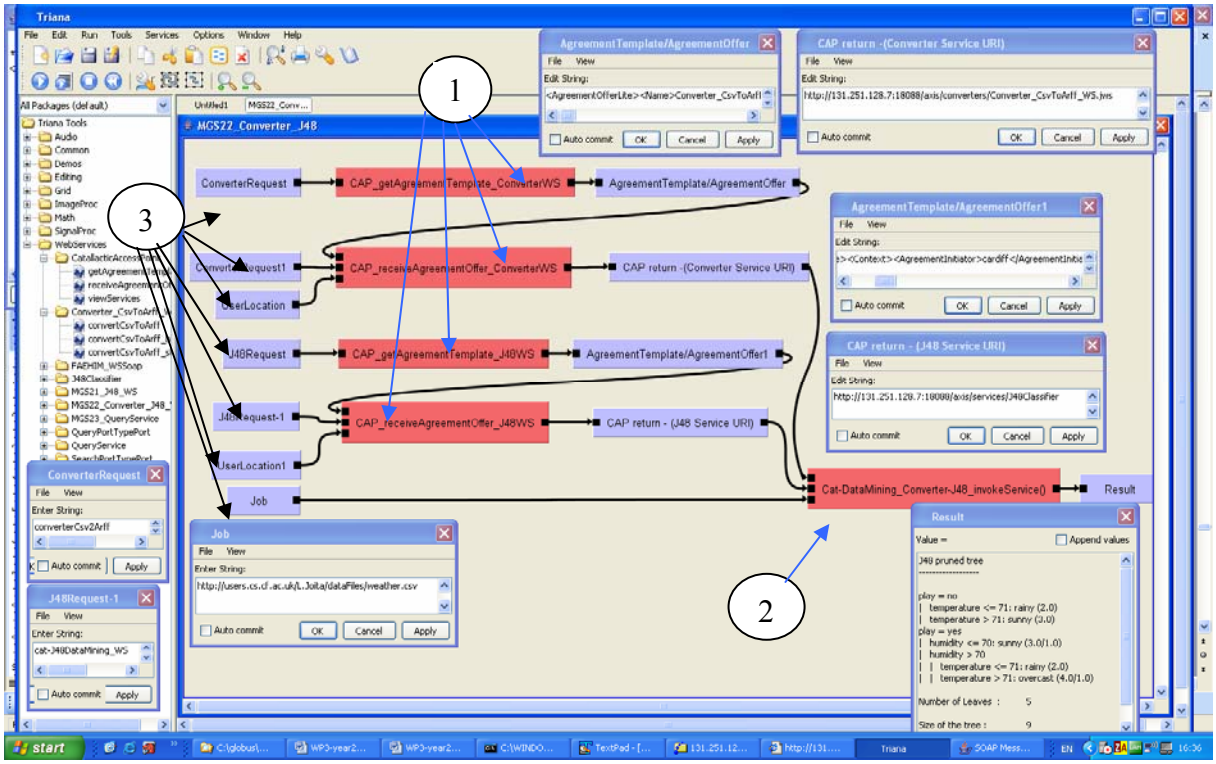

Figure 7.3 – Cat-DataMining -- interactions between MGS, CAP and Catallactic middleware (GMM)

... <wsa:To soapenv:mustUnderstand="0">
http://131.251.128.7:18088/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
</wsa:To> ...

Listing 7.14 – CAP invocation – for the Converter service

... <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
*<AgreementTemplateLite><Name>**Converter_CsvToArff-ComplexService**</Name><Context><AgreementInitiator></AgreementInitiator><StartingTime></StartingTime><TerminationTime></TerminationTime></Context><Terms><BasicServiceType>Converter_CsvToArff</BasicServiceType><NumberOfBasicServiceNodes><!-- between 1 to 10 --></NumberOfBasicServiceNodes><BasicServiceConstraints><ResponseTimePerRequest>10<!-- maximum milliseconds --></ResponseTimePerRequest></BasicServiceConstraints><PayForService></PayForService></Terms></AgreementTemplateLite>*
</ns1:getAgreementTemplateReturn>

Listing 7.15 – Application and CAP interaction – agreement template, for Converter Service, is uploaded by the application instance

... <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
*<AgreementTemplateLite><Name>**J48-ComplexService**</Name><Context><AgreementInitiator></AgreementInitiator><StartingTime></StartingTime><TerminationTime></TerminationTime></Context><Terms><BasicServiceType>J48</BasicServiceType><NumberOfBasicServiceNodes><!-- between 1 to 10 --></NumberOfBasicServiceNodes><BasicServiceConstraints><ResponseTimePerRequest>10<!-- maximum milliseconds --></ResponseTimePerRequest></BasicServiceConstraints><PayForService></PayForService></Terms></AgreementTemplateLite>*
</ns1:getAgreementTemplateReturn>...

Listing 7.16 – Application and CAP interaction – agreement template, for J48 Service, is uploaded by the application instance

```
... <agreementOffer xsi:type="xsd:string">
    &lt;AgreementOfferLite&gt;&lt;Name&gt;Converter_CsvToArff-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;cardiff&lt;/AgreementInitiator&
gt;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;&lt;/Cont
ext&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;Converter_CsvToArff&lt;/BasicServiceType&gt;&lt;Number
OfBasicServiceNodes&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimePerRequest&
gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;80&lt;/PayF
orService&gt;&lt;/Terms&gt;&lt;/AgreemenOfferLite&gt;
    </agreementOffer> ...
```

Listing 7.17 – Application and CAP interaction – agreement offer, for Converter service, is sent by the application instance

```
...  <ns1:receiveAgreementOfferReturn xsi:type="xsd:string">
     http://131.251.128.7:18088/axis/converters/Converter_CsvToArff_WS.jws
    </ns1:receiveAgreementOfferReturn> ...
```

Listing 7.18 – A decision is return by CAP – the EPR of the Converter service

```
...  <agreementOffer xsi:type="xsd:string">
    &lt;AgreementOfferLite&gt;&lt;Name&gt;J48-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;cardiff&lt;/AgreementInitiator&
gt;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;&lt;/Cont
ext&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;J48&lt;/BasicServiceType&gt;&lt;NumberOfBasicServiceNo
des&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimePerRequest&
gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;70&lt;/PayF
orService&gt;&lt;/Terms&gt;&lt;/AgreementOfferLite&gt;
    </agreementOffer> ...
```

Listing 7.19 – Application and CAP interaction – agreement offer, for J48 service, is sent by the application instance

```
...  <ns1:receiveAgreementOfferReturn xsi:type="xsd:string">
     http://131.251.128.7:18088/axis/services/J48Classifier
    </ns1:receiveAgreementOfferReturn> ...
```

Listing 7.20 – A decision is return by CAP – the EPR of the J48 service

```
... <ns1:invokeService soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
    <serviceURI_Converter xsi:type="xsd:string">
     http://131.251.128.7:18088/axis/converters/Converter_CsvToArff_WS.jws
    </serviceURI_Converter>
    <serviceURI_J48 xsi:type="xsd:string">
     http://131.251.128.7:18088/axis/services/J48Classifier
    </serviceURI_J48>
    <job xsi:type="xsd:string">
     http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.csv
    </job>
  </ns1:invokeService> ...
```

Listing 7.21 – Invocation of the workflow (Converter and J48 services) and job submission

```
...      <ns1:invokeServiceReturn xsi:type="xsd:string">
        J48 pruned tree
------------------
play = no
|   temperature &lt;= 71: rainy (2.0)
|   temperature &gt; 71: sunny (3.0)
play = yes
|   humidity &lt;= 70: sunny (3.0/1.0)
|   humidity &gt; 70
|   |   temperature &lt;= 71: rainy (2.0)
|   |   temperature &gt; 71: overcast (4.0/1.0)
Number of Leaves  :     5
Size of the tree :      9
      </ns1:invokeServiceReturn>  ...
```

Listing 7.22 – Result return to the user, encoded in the SOAP message

7.2 Alternative P2P overlays for Middleware

We have been using JXTA as P2P toolkit for the middleware to provide two fundamental functions: multicast and decentralized query between agents (multicast with response). For that means, we use the JXTA Resolver Service, a low level protocol which allows the implementation of custom handlers to process the messages on each node. This gives us the required flexibility to implement the query processing, but still relies in Jxta to propagate the messages.

An important drawback with JXTA Resolver Service is that, at least for the experience we gained during its usage, it is extraordinarily hard to get reliable query answering on the internet. We have experienced loss of messages, unexpected query handling problems which made impossible for us to deploy the middleware on the internet. The only feasible manner to work with JXTA Resolver reliably has been for us deploying the agents on a subnet and use multicast.

We tested JXTA setting a set of nodes configured as relays (that its, capable of propagating messages across sub-net boundaries to other known relays) connected thought direct links into a complex topology, as shown in Figure 7.4. Our findings are that, the Resolver Service is not behaving as expected. We would expect the messages to be delivered to a subset of the peers, as long as the time to live (number of hops to be follow) were not reached. However, the actual implementation of the protocol has some limitations in this kind of setups. First, messages are propagated to only one of the know relay peers, not to all of them, to avoid message flooding. Second, messages that arrive to a relay peer are not re-propagated to other relays, but only to other peers in the local subnet. Message handlers are then responsible to re-propagate the message if required. These limitations were confirmed by developers of the Jxta platform.

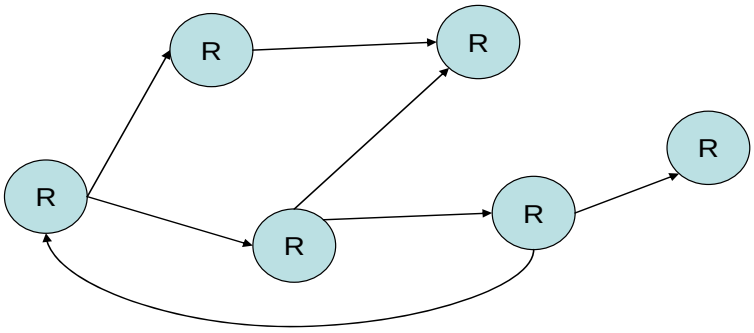

Figure 7.4 - Sample test topology

We propose two potential solutions to solve this issue with the JXTA Resolver:

- Modify the implementation of the Resolver Protocol to overcome the limitations we have identified. We have risen this issue to the community and there is a general understanding of the need for an enhancement. However, there is not a clear commitment to undertake the changes in a shot period. We have also contacted other projects facing similar problems with Jxta and we are evaluating the feasibility, within the limits of the CATNETS project, of addressing the required modifications together.

- We evaluate the possibility of switching to another P2P toolkit. We have survey some existing toolkits which offer the two basic functions required by the middleware. We have identified some requirements that a replacement should comply to be considered:

- Be available under an open source license
- Have a fully functional implementation
- Be compatible with the architecture of the middleware
- Count with the support of a group or a well established community.
- Be under active development that might incorporate state of the art approaches
- Be easy to use and well documented

Following this criteria, a few options are available:

- DKS (http://dks.sics.se), developed by the Swedish Institute of Computer Science (SICS)
- Bamboo (http://bamboo-dht.org), developed originally at the Berkley university and now an open source project
- FreePastry (http://freepastry.org/), developed by Rice University

We have identified DKS as the more promising option not only for its technical merits, but also due to the interest of the DKS group to support the integration effort and the potential synergy between our research groups in the context other European projects, which might lead to a long term collaboration .

We have initiated a proof of concept to test the feasibility of the integration and assess the ease of use and the maturity of the tool. Based on these results we will decide whether to continue with DKS or evaluate other options.

8 Relation to other WPs

8.1 WP1

The objective of WP1 (Theoretical and Computational Basis) is the conceptual design of market mechanisms for the CATNETS scenario. This includes the design of auctions (denoted as central case) and a decentralized bargaining strategy (denoted as Catallactic case). The Catallatic case is implemented as a resource allocation mechanism in the prototype.

The relations to WP3 are as follows:

- The decentralized bargaining strategy is implemented in the middleware enabling the analysis of the strategy behaviour in a real prototype environment. Scenarios for using the strategy were developed together with WP3. These scenarios describe interaction pattern of the prototype environment with the bargaining strategy integration in the middleware. This includes the specification interfaces for messaging and external interfaces to customized plug-ins (plug-in for mapping basic service demand to a resource bundle request and plug-in for local resource managers providing information about the current status of the resource).

- Based on an example scenario showing the basic middleware functionality, the software agents are developed and integrated into the middleware. The detailed description of interaction pattern of the software agents and the developed components presents chapter 5, deliverable WP 1. Results of the integration testing constitute further optimization issues of the strategy in year 3.

- The central cases (auctions) are not implemented in the prototype. We profit, however, from the knowledge provided by WP3: We see what *can* be realized in a price-based resource allocation mechanism, what is conceptually *required* for a successful application of markets in ALN's, and we learn which theoretical concepts are *not implementable* from an application point-of-view. For instance, we have designed a market language (i.e. how agents formulate their bids) by means of WS-Agreement to ensure compatibility with the prototype and common GGF standards. The lessons learnt by WP3 will further influence the refinement of our auction mechanisms in the third project year.

8.2 WP2

The objective of WP2 (Simulator) is the design and implementation of a simulation tool to be used for the evaluation of the central and catallactic economic allocation mechanisms. As described in deliverable D4.1, simulations are supposed to produce a set of technical metrics which are combined in order to obtain the evaluation of the allocation mechanisms in terms of economic parameters.

The relations to WP3 are as follows:

- The simulator embeds into a single framework both the central and catallactic allocation mechanisms while the prototype embeds only the catallactic mechanism. The reason for this is that the prototype is supposed to demonstrate the feasibility of real-world ALN applications based on the catallactic service/resource allocation mechanism, while the goal of the simulator development is to produce a tool which permits to run a large set of

scenarios for an effective and extensive comparison of the performance of the two allocation mechanisms.

- As the CATNETS simulator directly derive from the grid simulator OptorSim, the implementation of the agent communication layer differs from the implementation of the corresponding middleware in the prototype. This implies some differences in the way the technical metrics are collected and how the catallactic strategy is implemented in the two tools. However, an effort has been made in order to minimise these differences so that the results obtained by the two tools are comparable. In the third year of the project an investigation of their actual comparability has to be performed. As a precondition, a set of simulation scenarios reflecting the operative condition of the prototype application has to be defined.

8.3 WP4

The objective of WP4 (Performance Evaluation) is the evaluation of market mechanisms for the CATNETS scenario. WP4 has design an evaluation framework for application to both simulator and prototype. The evaluation framework is composed of technical and economic metrics specification, analysis procedures and evaluation in economic and technical terms.

The relations to WP3 are as follows:

- Measurement mechanisms must be implemented in the prototype to gather metrics specify in the evaluation framework: a generic measurement framework has been design, it is describe in chapter 2.2 in Deliverable WP4.2. The implementation of data collectors has show that some metrics can not be measured directly by the GMM, but other similar metrics can be obtained. Those metrics will be used unless they constitute a big drawback.

- Preliminary test on the prototype have reported some performance data that has been used to test the analysis procedures and obtain some economic metrics. This test evaluation is described in chapter 3 in Deliverable WP4.2.

- Prototype and simulator evaluation results should be compared to ascertain that simulated scenarios performance results are similar to real prototype results.

9 Related Work and Efforts

9.1 GGF, W3C and OASIS

The work being undertaken in this WP is making direct use of standards being developed in the W3C, Global Grid Forum (now called the "Open Grid Forum") and OASIS. The use of the standards approach will ensure that components within the GMM, or the components of the application prototypes, may be directly used within other application scenarios (not associated with this project). The following standards are being adopted in the project:

Web Services (SOAP, WSDL): These standards are used to define interfaces to services (such as the data mining services or the data converter services), and encode the interaction between services. A number of examples of SOAP messages exchanged between the services have been provided in section 7.

WS-Agreement: although not yet fully released as a standard, the WS-Agreement specification is used to encode service requests (contracts) between the application and the access point to the market (referred to as the Catallactic Access Point).

9.2 GRAAP Working Group

The need for a Service Level Agreement (SLA) has been outlined by a number of Grid application users – as part of work that has been undertaken within the Global Grid Forum GRAAP working group. Some have identified such SLAs to be static descriptions of the capabilities of resource providers, whilst others advocate a more dynamic approach which involves negotiation of SLAs. As Grid infrastructure moves towards commercial adoption, the use of SLAs will become more significant. The GRAAP working group released the WS-Agreement specification on September 20, 2005 as a "*GWD-R" (a Grid Forum Working Document)*. This specification has already undergone an extensive review process (via the GRAAP mailing list and telecon. sessions). The current version of the WS-Agreement specification provides a collection of terms that may be used to specify an agreement, but does not identify any particular set of terms that identify what service properties are to be delivered under the agreement. Currently, WS-Agreement involves defining an SLA statically, and does not provide any support for negotiation. One of the reasons for these choices is to keep the specification simple to use and deploy. Negotiation is therefore seen as an operation that could be used outside the specification – i.e. a negotiation could result in the formation of a WS-Agreement template, and not used to modify the template once it has been created.

The limitations with a static agreement have been recognized by the WS-Agreement community, and significant effort has recently been invested in describing dynamic agreements. This may involve two aspects:

- The need to modify an agreement that had already been established -- especially if the agreement is used at a time much later than when the agreement had been defined. The requirement here relates to comparing the cost of re-establishing a new agreement vs. being able to adapt an agreement that is already in place.

- The need to support flexibility in the agreement if an agreement initiator is not fully aware of the operating environment when the agreement is defined. In this case, the agreement initiator may not have enough information to determine what to ask for

from a provider. This is likely to be the case when an agreement initiator or provider operates with imprecise knowledge about the other party involved in the agreement.

Based on these requirements, two types of dynamic agreements may be defined:

- Case 1: Static Agreement. In this case, it was necessary to identify Service Description Terms, Guarantee Terms, and Service Level Objectives (SLOs). Both Guarantee terms and SLOs were to be precisely defined at agreement creation time.

- Case 2: Dynamic Agreement. In this case, it was necessary to identify Service Description Terms, Guarantee Terms which were now defined as ranges or as functions, and Service Level Objectives which were defined as ranges or as functions. The use of range-based or function-based agreements provided a useful basis for supporting dynamicity in the agreement.

WS-Agreement has been used by Toshiyuki Nakata at NEC (Japan), for instance, to support multiple SLAs within an Internet Service Provider (ISP), and requirements of being able to combine terms in multiple SLAs to create an aggregate SLA. He has also identified the need to define common terms within the SLA. Within the CATNETs project, WS-Agreement is being used as the basis for choosing service providers in a Grid market. The terms we use in the WS-Agreement are a combination of application specific terms (such as a data mining service or a query service), along with economic terms (such as price and utility) that are being defined in other work packages in the project.

Members of the CATNETs project have continued to participate in WS-Agreement efforts, and have provided application examples – developed in this WP (and in collaboration with other WP leaders).

10 Conclusion

10.1 Catallactic-based Grid Middleware

Functional testing results, as well as first experiments evaluating the performance of a Catallactic Grid middleware implementation have been presented. The results so far are encouraging. Two important benefits have been derived from this process: First, important refinements on the GMM in all levels, as well as useful clarification on which future steps need to be taken in order to complete the development successfully by the end of the project. Second, several convenient tools have been identified and tested which have been (an will be further used) to in-deep evaluate the GMM.

The GMM continues presenting a flexible service selection tool to be applied in Grid, P2P and CDN domains. This meets current trends in Grid Computing, showing tendency towards the use of a service oriented approach, along with support for resource virtualization – these provide the main drivers for an increasing integration of the resource negotiation mechanisms offered by our middleware with the base platforms presented (such as GT4).

It is important to stress the exclusive role of the middleware as a service/resource discovery and selection tool. From the amenability point of view (broader application of the middleware to other grid, p2p and CDN applications) it is required a very low coupling with the application. The middleware should never be providing application specific services as this would decrease modularity and complicate any further usage in different scenarios.

Providing interaction with the Catallactic market via an access point (the CAP), we ensure that the application does not need to be modified internally, hence complying with the requirements from he paragraph above. In addition, the peer-2-peer nature of the Catallactic market model (as opposed to other market models proposed in Grid computing that make use of centralized brokers) make our approach more suitable to distributed decentralized systems, such as computational and data Grids.

Next steps include the full development of measurement components to support the middleware, and to assess the performance of the developed application. Also, we envision proving the GMM in the new richer application models provided by CU.

10.2 Applications using Catallactic-based middleware

The implementation of two prototypes with a Catallactic based mechanism, as well as a generalisation of application with the GMM, have been presented and have raised interesting conclusions outlined below:

- COVITE prototype has been designed and developed as a centralised system, where all services and resources are under one authority control [COVITE04]. All VOs created via the central system use the resources of the authority. A decentralised solution, such as the proposed Cat-COVITE prototype, will create great benefits to such central systems, as the demand of services and resources from VOs will be better solved via the CATNETS markets.

- A second application instance based on data mining mechanisms – Cat-DataMining, has been developed and discussed in this report. One main problem that data mining services are addressing is of data that is typically too inconsistent and difficult to

understand into such forms that are more compact, useful and understandable. This can be achieved via specific data-mining methods for pattern discovery and extraction. This process can be structured into a discovery pipeline/workflow, involving access, integration and analysis of data from disparate sources, and to use data patterns and models generated through intermediate stages. The GMM addresses the need of such data mining services to be found just-in-time and used by the application services.

- The CATNETS markets expect to provide VOs the possibility to gain profits out of their spare resources and services.

- The Catallactic mechanism expects to help systems in discovering and selecting resources and services on demand and just in time, as application processes can make use of third parties services or can demand more and more resources.
- Metrics relevant both to the middleware and the application instances have been elaborated and experiments and prototype tests have been done. The process of extracting and analysing the results are shown in Deliverable WP4.

10.3 Open issues and future work

The work on integrating Cat-COVITE and Cat-DataMining prototype application instances and Catallactic middleware has been accomplished and a proof-of-concept has been developed using Triana. Several remaining issues will be targeted in the following months, following this roadmap:

- Further integration in Triana of use cases shown in section 3.
- Complete integration and functioning tests.
- More tests and experiments to be done, while extracting metrics to be compared with the simulator case.
- Refinement in measurement components to support the middleware

References

[ADAR00] E. Adar and B. A. Huberman – "Free riding on Gnutella", First Monday, 5(10), 2000.

[ALI05] A.S. Ali, O.F. Rana, I.J. Taylor – "Web Services Composition for Distributed Data Mining", ICPP 2005 Workshops, International Conference on Parallel Processing, 14-17 June 2005, pp. 11-18

[BAVIER04] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, Operating System Support for Planetary-Scale Network Services. First Symposium on Networked Systems Design and Implementation (NSDI) (March 2004), 253-266

[BUYYA00] R. Buyya and J. Abramson, D.and Giddy – "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid", In Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), 2000.

[CatCOVITE05] Liviu Joita, Omer F. Rana, Pablo Chacin, Isaac Chao, Felix Freitag, Leandro Navarro, Oscar Ardaiz (2005) - "Application Deployment using Catallactic Grid Middleware", Proceedings of the 3rd International Workshop on Middleware for Grid Computing (MGC 2005), co-located with ACM/USENIX/IFIP Middleware 2005, 28 November - 2 December 2005, Grenoble, France.

[CHACIN06] Pablo Chacin, Liviu Joita, Björn Schnizler - "Flexible Architecture for Supporting Auctions in Grids", Proceedings of the 2nd International Workshop On Smart Grid Technologies 2006 (SGT2006) Workshop, In co-location with the 3rd International Conference on Autonomic Computer, 12-16 June 2006, Dublin, Ireland.

[CHAPIN99] S. J. Chapin, D. Katramatos, J. Karpovich, and A. S. Grimshaw – "The legion resource management system", In Dror G. Feitelson and Larry Rudolph, editor, Job Scheduling Strategies for Parallel Processing, pages 162–178. Springer Verlag, 1999.

[COHEN03] Bram Cohen, Incentives Build Robustness in BitTorrent 2003

[COVITE04] Joita L., Pahwa J. S., Burnap P., Gray A., Rana O., and Miles J. Supporting Collaborative Virtual Organisations in the Construction Industry via the Grid. Proceedings of the UK e-Science All Hands Meeting 2004, 31st Aug.-3rd Sept. 2004 Nottingham, UK.

[ECLIPSE06] www.eclipse.org/

[EYMANN05] T. Eymann, O. Ardaiz, M. Catalano, P. Chacin, I. Chao, F. Freitag, M. Gallegati, G. Giulioni, L. Joita, L. Navarro, D. Neumann, O. Rana, M. Reinicke, R. C. Schiaffino, B. Schnizler,W. Streitberger, D. Veit, and F. Zini - "Catallaxy based Grid markets", Proceedings of the First International Workshop on Smart Grid Technologies (SGT05), 2005.

[FOSTER97] I. Foster and C. Kesselman – "Globus: A metacomputing infrastructure toolkit", The International Journal of Supercomputer Applications and High Performance Computing, 11(2):115–128, 1997.

[FREEDMAN04] Michael J. Freedman, Eric Freudenthal, and David Mazières, Democratizing Content Publication with Coral,. In Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04) San Francisco, CA, March 2004

[JMETER06] jakarta.apache.org/**jmeter**/

[MONTRESOR03] A. Montresor and O. Baboglu – "Biology-inspired approaches to peer-to-peer computing in bison", In Proceedings of the 3rd International Conference on Intelligent System Design and Applications (ISDA'03), Advances in Soft Computing, pages 515–522. Springer-Verlag, Aug. 2003.

[QUINLAN93] R. Quinlan – "C4.5: Programs for Machine Learning", Morgan Kaufmann Publishers, San Mateo, CA, 1993

[WSAG05] Web Services Agreement Specification (WS-Agreement), 14 June 2006, Grid Resource Allocation Agreement Protocol WG (GRAAP-WG). Available at (accessed on 30 June 2006): https://forge.gridforum.org/sf/docman/do/listDocuments/projects.graap-wg/docman.root.current_drafts

[PREIST98] C. Preist, M. van Tol, Adaptive agents in a persistent shout double auction. In Proceedings of the First international Conference on information and Computation Economies (Charleston, South Carolina, United States, October 25 - 28, 1998). ICE '98. ACM Press, New York, NY, 11-18.

[SUN] http://www.sun.com; http://www.network.com

[YEO04]C.S. Yeo, R. Buyya, Pricing for utility-driven resource management and allocation in clusters. Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004), Ahmedabad, India, December 2004. Allied Publisher: New Delhi, India; 32–41.

[WHW+04] S. White, J. Hanson, I. Whalley, D. Chess, J. Kephart (2004),"An Architectural Approach to Autonomic Computing", International Conference on Autonomic Computing

[WP305] Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freigtag, Liviu Joita, Manuel Medina, Leandro Navarro, Omer F. Rana, Miguel Valero (2005), Deliverable 3.1: Implementation of additional services for the economic enhanced platforms in Grid/P2P platform: Preparation of the concepts and mechanisms for implementation

[WST06] www.eclipse.org/webtools/wst/main.html

Annex A

A.1 GMM scripts

lauchMiddleware.sh

```
#!/bin/sh
# script to lauch middleware on the remote machine

#USAGE:
# launchMiddleware start arguments
# first starts tomcat and deploy Web Service
# then launches middleware on Server or Client mode (depends on given
config file)
# middleware proccess  PID is stored in a file

#launchMiddleware stop
# stop middleware and shutdown tomcat
# removes PID file from current middleware execution

#lauchMiddleware status
# check if middleware is running or not

#Variables
YOURPATH=$HOME/CatnetsMiddlewareA2LN
CATALINA=apache-tomcat-5.5.12
CATALINA_HOME=$YOURPATH/$CATALINA
SERVICE_CLASSES_PATH=$CATALINA_HOME/webapps/axis/WEB-
INF/classes/faehimWService/classifiers
DEPLOY_DESCRIPTOR_SERVICE=deploy_J48.wsdd
TMP=/tmp
PIDFILE=${TMP}/catnets_pidfile.pid

#Environment Variables
#JAVA
export JAVA_HOME=$YOURPATH/java
#export JAVA_HOME=/usr/java/jre1.5.0_06/
export PATH=$JAVA_HOME/bin:$PATH
#TOMCAT
export CATALINA_HOME=$YOURPATH/apache-tomcat-5.5.12
export PATH=$CATALINA_HOME/bin:$PATH
#AXIS
export AXIS_HOME=$YOURPATH/axis-1_3
export AXIS_LIB=$AXIS_HOME/lib
export AXIS_CLASSPATH=$AXIS_LIB/axis.jar:$AXIS_LIB/axis-
ant.jar:$AXIS_LIB/axis-schema.jar:$AXIS_LIB/commons-discovery-
0.2.jar:$AXIS_LIB/commons-logging-
1.0.\4.\jar:$AXIS_LIB/jaxrpc.jar:$AXIS_LIB/saaj.jar:$AXIS_LIB/log4j-
1.2.8.jar:$AXIS_LIB/wsdl4j-1-5-1.jar
export
CLASSPATH=$CLASSPATH:.:$AXIS_CLASSPATH:$CATALINA_HOME/webapps/axis/WEB-
INF/lib/activation.jar:$CATALINA_HOME/webapps/axis/WEB-
INF/lib/mail.jar:$YOURPATH/J48_WS_Client.class
export TERM=xterm

# rsync
ORIGEN=$2;
ROLE=$3;

#Arguments
```

```
ARGUMENT1=$4 #confif file identifier
ARGUMENT2=$5 #config file
ARGUMENT3=$6 #node name identifier
ARGUMENT4=$7 #node name
# just for simulated load scenarios
ARGUMENT5=$8 #agent.resource.param.utilizationAverage
ARGUMENT6=$9 # resource utilization

start(){

#stop any previous java proccess running (any tomcat or middleware
instance)
stop;

# sincronize with code base
echo "syncrhronizing with rsync"
 #rsync -vaz --delete --recursive --force $ORIGEN:$YOURPATH $HOME
echo "done sync"

#check CLASSPAHT
echo "CLASSPATH=$CLASSPATH"

#configuring client or server
#if server
if test $ROLE = "BS"; then
echo "launching a server"
cp $YOURPATH/middleware_BS.jar $YOURPATH/middleware/middleware.jar
cp $YOURPATH/BSHostingConfig.properties $YOURPATH/middleware/
#deploying J48 service
echo "deploying J48 into Tomcat"
# start tomcat
$CATALINA_HOME/bin/startup.sh
#sleep to make sure the adminClient is up
sleep 3
# deploy service
java  org.apache.axis.client.AdminClient
$SERVICE_CLASSES_PATH/$DEPLOY_DESCRIPTOR_SERVICE
#sleep to wait for service deployment
sleep 3

# else client
else
echo "launching a client"
cp $YOURPATH/middleware_CS.jar $YOURPATH/middleware/middleware.jar
cp $YOURPATH/CSHostingConfig.properties $YOURPATH/middleware/
#delete previous execution files and renew
rm -f  $YOURPATH/executionTime/*.*
mkdir -p  $YOURPATH/executionTime
fi
#common
cp $YOURPATH/profile.xml $YOURPATH/middleware/;

#START MIDDLEWARE ITSELF

if ! [ -f $PIDFILE ]; then
#JAVA
echo "setting environment variables"
export PATH=$YOURPATH/middleware.jar:$PATH
echo "Arguments for middleware: $ARGUMENT1 $ARGUMENT2 $ARGUMENT3
$ARGUMENT4"
#launch ServiceProvider app
```

```
java -jar middleware/middleware.jar $ARGUMENT1 $ARGUMENT2 $ARGUMENT3
$ARGUMENT4 $ARGUMENT5 $ARGUMENT6  &
PID=$!
echo "launching middleware with PID: " $PID
echo $PID > $PIDFILE
else
 echo "middleware already running, stop first the current instance"
fi
}

stop(){
if ! [ -f $PIDFILE ]; then
   echo "middleware not running, nothing to shutdown"
   # kill tomcat or other java processes runnning
   killall java
   #killall lookbusy
   killall a.out


else
killall java
echo "shutting down middleware..."
kill -TERM $(<$PIDFILE)
rm -f $PIDFILE
fi
}


status(){
if ! [ -f $PIDFILE ]; then
    echo "middleware not running"
elif [ -s $PIDFILE -a -d /proc/$(<$PIDFILE) ]; then
    echo "middleware running";
else
    echo "stale pidfile"
    rm -f $PIDFILE
fi
}

case "$1" in
start)
      start;;
stop)
      stop;;
status)
      status;;
*)
      echo $"Usage: launchMiddleware {start [arguments]|stop|status}"
      exit 1
esac
```

---

deployMiddleware.sh

```
#!bin/sh
#deployMiddleware.sh
# Deploy in a list of nodes all the components for middleware execution:
# in all nodes: Tomcat, Axis, middleware launch scripts
# in server nodes: Web/Grid services, middleware jar as server
# in client nodes: Client, middleware jar as client
```

```
#Variables
YOURPATH=$HOME/CatnetsMiddlewareA2LN
REPOSITORY=$YOURPATH
CATALINA=apache-tomcat-5.5.12
CATALINA_HOME=$YOURPATH/$CATALINA
AXIS=axis-1_3
AXIS_HOME=$YOURPATH/$AXIS
JAVA_HOME=$YOURPATH/java

# clean directory structure in remote node

#./vxargs -a nodeList.txt  ssh {} rm -r -f $YOURPATH
./vxargs -a nodeList.txt  ssh {} mkdir -p $YOURPATH
./vxargs -a nodeListCS.txt  ssh {} rm -r -f $YOURPATH/executionTime
./vxargs -a nodeList.txt  ssh {} rm -r -f $YOURPATH/middleware
./vxargs -a nodeList.txt  ssh {} mkdir -p $YOURPATH/middleware
./vxargs -a nodeListCS.txt  ssh {} mkdir -p $YOURPATH/executionTime
./vxargs -a nodeList.txt  ssh {} rm -r -f $CATALINA_HOME
./vxargs -a nodeList.txt  ssh {} rm -r -f $AXIS_HOME
./vxargs -a nodeList.txt  ssh {} rm -r -f $JAVA_HOME
#./vxargs -a nodeList.txt ssh {} rm -f -r $YOURPATH/*

# install own java in remote node
./vxargs -a nodeList.txt scp -r  $REPOSITORY/java {}:$YOURPATH

#install tomcat and axis in remote node
./vxargs -a nodeList.txt scp $REPOSITORY/apache-tomcat-5.5.12.zip
{}:$YOURPATH
./vxargs -a nodeList.txt scp $REPOSITORY/axis-bin-1_3.zip  {}:$YOURPATH
./vxargs -a nodeList.txt  ssh {} "cd $YOURPATH; unzip apache-tomcat-
5.5.12.zip"
./vxargs -a nodeList.txt  ssh {} "cd $YOURPATH; unzip axis-bin-1_3.zip"
./vxargs -a nodeList.txt  ssh {} cp -r $AXIS_HOME/webapps/axis
$CATALINA_HOME/webapps/axis

# copy J48 Web Service deployment data and execution dataset
# copy data set to tomcat root
./vxargs -a nodeListBS.txt scp $REPOSITORY/aaaa.arff
{}:$CATALINA_HOME/webapps/ROOT/J48_WS_Data.arff

# copy J_48 data into corresponde tomcat/axis directory
./vxargs -a nodeList.txt scp -r  $REPOSITORY/faehimWService
{}:$CATALINA_HOME/webapps/axis/WEB-INF/classes/

#copy Client class into Client nodes
./vxargs -a nodeListCS.txt scp $REPOSITORY/J48_WS_Client.class {}:$YOURPATH
#./vxargs -a nodeListCS.txt scp $REPOSITORY/J48_WS_Client.class
{}:$YOURPATH/middleware

#copy libs required for Web Services deployment and invocation
./vxargs -a nodeList.txt scp $REPOSITORY/activation.jar
{}:$CATALINA_HOME/webapps/axis/WEB-INF/lib
./vxargs -a nodeList.txt scp $REPOSITORY/mail.jar
{}:$CATALINA_HOME/webapps/axis/WEB-INF/lib
./vxargs -a nodeList.txt scp $REPOSITORY/weka.jar
{}:$CATALINA_HOME/webapps/axis/WEB-INF/lib


# copy middleware launching scrpt, middleware jar, middleware config file,
middleware required libraries and jxta profile
./vxargs -a nodeList.txt scp $REPOSITORY/launchMiddleware.sh {}:$YOURPATH/
```

```
./vxargs -a nodeList.txt scp -r $REPOSITORY/middleware/lib
{}:$YOURPATH/middleware/lib
./vxargs -a nodeList.txt scp $REPOSITORY/profile.xml {}:$YOURPATH
./vxargs -a nodeListBS.txt scp $REPOSITORY/middleware_BS.jar {}:$YOURPATH
./vxargs -a nodeListCS.txt scp $REPOSITORY/middleware_CS.jar {}:$YOURPATH
./vxargs -a nodeListBS.txt scp $REPOSITORY/BSHostingConfig.properties
{}:$YOURPATH/BSHostingConfig.properties
./vxargs -a nodeListCS.txt scp $REPOSITORY/CSHostingConfig.properties
{}:$YOURPATH/CSHostingConfig.properties

#permissions
./vxargs -a nodeList.txt ssh {} "cd $YOURPATH; chmod -R +xwr *"
```

---

collectMetrics.sh

```
#!/bin/sh
#collectMetrics.sh
#collects metrics and logs from the middleware in a set of machines listed
in nodeListBS.txt and nodeListCS.txt
#metrics and logs from an experiment are stored in a file
metricAndLogsMiddleware_DATE

YOURPATH=$HOME/CatnetsMiddlewareA2LN
RESULTSPATH=$HOME/results
#variables
LABEL=$(date +%Y-%m-%d-%H-%M)

#erase results from previous collection
rm -rf $RESULTSPATH/metrics
rm -rf $RESULTSPATH/logs

# create directories to collect metrics
mkdir $RESULTSPATH/metrics
mkdir $RESULTSPATH/logs
mkdir $RESULTSPATH/metrics/price
mkdir $RESULTSPATH/metrics/utilization
mkdir $RESULTSPATH/metrics/sellSuccess
mkdir $RESULTSPATH/metrics/buySuccess
mkdir $RESULTSPATH/metrics/bid
mkdir $RESULTSPATH/metrics/negotiationTime
mkdir $RESULTSPATH/metrics/executionTime

# use vxargs to collect remotelly the metrics and logs from the remote
nodes (BSs and CSs)

./vxargs -a nodeListBS.txt scp {}:$YOURPATH/middleware/price.txt
$RESULTSPATH/metrics/price/BS{}_price.txt
cat $RESULTSPATH/metrics/price/* >> $RESULTSPATH/metrics/price/price.txt

./vxargs -a nodeListBS.txt scp {}:$YOURPATH/middleware/utilization.txt
$RESULTSPATH/metrics/utilization/BS{}_utilization.txt
cat $RESULTSPATH/metrics/utilization/* >>
$RESULTSPATH/metrics/utilization/utilization.txt

./vxargs -a nodeListBS.txt scp {}:$YOURPATH/middleware/bid.txt
$RESULTSPATH/metrics/bid/BS{}_bid.txt
cat $RESULTSPATH/metrics/bid/* >> $RESULTSPATH/metrics/bid/bid.txt

./vxargs -a nodeListBS.txt scp {}:$YOURPATH/middleware/sellSuccess.txt
$RESULTSPATH/metrics/sellSuccess/BS{}_.txt
```

```
cat $RESULTSPATH/metrics/sellSuccess/* >>
$RESULTSPATH/metrics/sellSuccess/sellSuccess.txt

./vxargs -a nodeListCS.txt scp {}:$YOURPATH/middleware/buySuccess.txt
$RESULTSPATH/metrics/buySuccess/CS{}_.txt
cat $RESULTSPATH/metrics/buySuccess/* >>
$RESULTSPATH/metrics/buySuccess/buySuccess.txt

./vxargs -a nodeListCS.txt scp {}:$YOURPATH/middleware/negotiationTime.txt
$RESULTSPATH/metrics/negotiationTime/CS{}_.txt
cat $RESULTSPATH/metrics/negotiationTime/* >>
$RESULTSPATH/metrics/negotiationTime/negotiationTime.txt

./vxargs -a nodeListCS.txt scp {}:$YOURPATH/executionTime/*
$RESULTSPATH/metrics/executionTime/
cat $RESULTSPATH/metrics/executionTime/* >>
$RESULTSPATH/metrics/executionTime/executionTime.txt

./vxargs -a nodeListBS.txt scp {}:$YOURPATH/middleware/log.txt
$RESULTSPATH/logs/log_BS{}.txt

./vxargs -a nodeListCS.txt scp {}:$YOURPATH/middleware/log.txt
$RESULTSPATH/logs/log_CS{}.txt

# store all metrcis and logs in a single directory
cp $RESULTSPATH/metrics/price/price.txt
$RESULTSPATH/metrics/utilization/utilization.txt
$RESULTSPATH/metrics/sellSuccess/sellSuccess.txt
$RESULTSPATH/metrics/buySuccess/buySuccess.txt
$RESULTSPATH/metrics/bid/bid.txt
$RESULTSPATH/metrics/negotiationTime/negotiationTime.txt
$RESULTSPATH/metrics/executionTime/executionTime.txt  $RESULTSPATH/metrics

# compress all metrics, logs and vxargs output  in a single tar file
```

tar -cvf $RESULTSPATH/metricsAndLogsMiddleware_$LABEL.tar
$RESULTSPATH/metrics $RESULTSPATH/logs $RESULTSPATH/outputVxargs

Annex B – CATNETS Application Repositories Settings

B.1 Cat-COVITE prototype

1. Supplier Database (SupplierDB) Design

Databases used to host the "supplier" products are: MySQL 5.0.20 and MS SQL Server 2000.

1.1 Use case of MySQL 5.0.20

Download MySQL from www.mysql.com and install it on your system. MySQL is running on both operating systems: Windows and Linux (various versions).

Steps to creating the "supplier" database:

- Connect to the MySQL server -- follow the details from the MySQL documentation.

- Create a database named SupplierDB.

```
mysql> CREATE DATABASE SupplierDB;
```

- Create a table named Product within the SupplierDB database.

```
mysql> use SupplierDB
mysql> CREATE TABLE Product (
      IDProduct SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
      ManufacturerName VARCHAR(50) NOT NULL,
      ProductName VARCHAR(50) NOT NULL,
      Price SMALLINT(5) UNSIGNED,
      PRIMARY KEY(IDProduct)
      );
```

- Fill in the table with some raw data.

```
mysql> INSERT INTO Product VALUES (NULL,'manufacturer1','chair1','10'),
(NULL,'manufacturer2','chair2','15'), (NULL,'manufacturer3','chair3','20'),
(NULL,'manufacturer4','chair4','25'), (NULL,'manufacturer5','chair5','30');
```

- Check the data entries.

```
mysql> SELECT * FROM SupplierDB.Product;
```

- As a "root" (administrator of the database) account, create user "catnets" with password "catnets" that has all privileges on the database SupplierDB. Or use the MySQL Adminitrator module to set up the privileges for user "catnets" to `SupplierDB` database.

```
mysql> GRANT all privileges on SupplierDB.* to 'catnets'@'localhost'
identified by 'catnets';
mysql> GRANT all privileges on SupplierDB.* to 'catnets'@'%' identified by
'catnets';
```

The first "catnets" account is for connection from the localhost, while the second "catnets" account is for connections from any host.

These accounts are for the MySQL security purposes. Usually the supplier databases can be accessed under an anonymous account. Therefore, all the queries that will run over the supplier databases will be under the "catnets" account.

2. Users' Loader Jobs Repository

The Loader database (Loader_DB), table UserLoaderJobs are used to load the initial parameters that emulate the clients' requirements. Within the context of COVITE project, a user of a formed Virtual Organisation for the duration of an architectural / engineering / construction project is interested in running query jobs to finding data needed within the project. Queries run to specific supplier databases that are register to supplying products that users are interested in.

Steps to creating the "Loader_DB" database:

Create a database named Loader_DB.

```
mysql> CREATE DATABASE Loader_DB;
```

Create a table named UserLoaderJobs.

```
mysql> use Loader_DB
mysql> CREATE TABLE UserLoaderJobs (
      ID SMALLINT(4) UNSIGNED NOT NULL AUTO_INCREMENT,
      Location VARCHAR(250) NOT NULL,
      ApplicationInstance VARCHAR(250) NOT NULL,
      Job TEXT NOT NULL,
      PRIMARY KEY(ID)
      );
```

Fill in the table `UserLoaderJobs` with the corresponding Cat-COVITE application instance data.

```
mysql> INSERT INTO Loader_DB.UserLoaderJobs VALUES (NULL,'cardiff','cat-
covite','SELECT * FROM SupplierDB.Product ORDER BY Price DESC;');
```

-- change `'cardiff'` with the relevant location for you. For example, enter `'barcelona'` or `'bayreuth'`.

- Check the data entries.

```
mysql> SELECT * FROM Loader_DB.UserLoaderJobs;
```

When launching the Cat-COVITE prototype instance, there is only need to provide two strings as inputs: "cat-covite" and "cardiff" (or your location).

When launching other application instances, there is only need to provide two strings as inputs: the name of the application instance and the location of the user.

- As a "root" (administrator of the database) account, create user "catnets" with password "catnets" that has all privileges on the database Loader_DB. Or use the MySQL Adminitrator module to set up the privileges for user "catnets" to `Loader_DB` database.

```
mysql> GRANT all privileges on Loader_DB.* to 'catnets'@'localhost'
identified by 'catnets';
mysql> GRANT all privileges on Loader_DB.* to 'catnets'@'%' identified by
'catnets';
```

3. Catallactic Access Point URLs Repository

The Catallactic Access Point (CAP) URLs repository represents a database that keeps track of available CAPs to be invoked by the Master Grid Service (MGS). The repository is hosted by a MySQL database.
The MGS will contact the wrapper Catallactic Access Point URLs Web Service (CAP_URLs_WS) to get the closest possible CAP URL located to the MGS. A number of CAPs are available to the application instances.

Steps to creating the CAP URLs repository:

- Create a database named CAP_URLs_DB.

```
mysql> CREATE DATABASE CAP_URLs_DB;
```

- Create a table named CAP_URLs within the CAP_URLs_DB database.

```
mysql> use CAP_URLs_DB
mysql> CREATE TABLE CAP_URLs (
      ID SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
      Location VARCHAR(50) NOT NULL,
      CAP_URL VARCHAR(250) NOT NULL,
      PRIMARY KEY(ID)
      );
```

- Fill in the table `CAP_URLs` with the corresponding CAP URLs site locations.

```
mysql> INSERT INTO CAP_URLs_DB.CAP_URLs VALUES
(NULL,'cardiff','http://131.251.47.197:8080/axis/CatallacticAccessPoint/Cat
allacticAccessPoint.jws');
```

-- change `'cardiff'` and `'http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws'` with the relevant locations for you. For example, enter `'barcelona'` and the relevant URL of the CAP in Barcelona.

- Check the data entries.

```
mysql> SELECT * FROM CAP_URLs_DB.CAP_URLs;
```

To update the CAP_URL within the CAP URLs repository:

```
mysql> UPDATE CAP_URLs_DB.CAP_URLs SET CAP_URL =
'http://131.251.128.7:18088/axis/CatallacticAccessPoint/CatallacticAccessPo
int.jws' WHERE Location = 'cardiff';
```
-- use your relevant updates data, if needed.

- As a "root" (administrator of the database) account, create user "catnets" with password "catnets" that has all privileges on the database CAP_URLs_DB. Or use the MySQL Adminitrator module to set up the privileges for user "catnets" to `CAP_URLs_DB` database.

```
mysql> GRANT all privileges on CAP_URLs_DB.* to 'catnets'@'localhost'
identified by 'catnets';
mysql> GRANT all privileges on CAP_URLs_DB.* to 'catnets'@'%' identified by
'catnets';
```

4. Catallactic Access Point Repository

The CAP repository represents each MySQL database linked to the individual CAP which is invoked by the MGS. This repository is hosted by a MySQL database and named CAP_AgreementTemplates_DB.

Steps to creating the CAP repository:

- Create a database named CAP_AgreementTemplates_DB.

```
mysql> CREATE DATABASE CAP_AgreementTemplates_DB;
```

- Create a table named AgreementTemplate within the CAP_AgreementTemplates_DB.

```
mysql> use CAP_AgreementTemplates_DB
mysql> CREATE TABLE `cap_agreementtemplates_db`.`agreementtemplate` (
        `ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        `ApplicationInstance` VARCHAR(250) NOT NULL DEFAULT '',
        `ServiceName` VARCHAR(200) NOT NULL DEFAULT '',
        `AgreementTemplateName` VARCHAR(200) NOT NULL DEFAULT '',
        `AgreementTemplate` TEXT NOT NULL DEFAULT '' COMMENT 'Agreement
Template Content',
        PRIMARY KEY(`ID`)
      )
      ENGINE = InnoDB
      COMMENT = 'Contains agreement templates of the complex services
available at this Catallactic Access Point location';
```

- Fill in the table with data:

```
mysql> INSERT INTO cap_agreementtemplates_db.agreementtemplate
VALUES(NULL,'cat-covite','QueryService','AT-
QueryService','<AgreementTemplateLite><Name>QueryComplexService</Name><Cont
ext><AgreementInitiator></AgreementInitiator><StartingTime></StartingTime><
TerminationTime></TerminationTime></Context><Terms><BasicServiceType>QueryS
ervice</BasicServiceType><NumberOfBasicServiceNodes>1 to 10<!-- between 1
to 10 --
></NumberOfBasicServiceNodes><BasicServiceConstraints><DBType>Architectural
/Engineering/Construction</DBType><ResponseTimePerRequest>10</ResponseTimeP
erRequest></BasicServiceConstraints><PayForService></PayForService></Terms>
</AgreementTemplateLite>');
```

Check that entry data is correct:

```
mysql> SELECT * FROM cap_agreementtemplates_db.agreementtemplate;
```

- Create a table named AgreementOfferReceived within the CAP_AgreementTemplates_DB.

```
mysql> CREATE TABLE `cap_agreementtemplates_db`.`AgreementOfferReceived` (
  `ID` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  `AgreementTemplateReference` SMALLINT UNSIGNED NOT NULL DEFAULT 0,
  `SenderName` VARCHAR(200) NOT NULL DEFAULT '',
  `AgreementOfferReceived` TEXT NOT NULL DEFAULT '',
  `Decision` TEXT NOT NULL DEFAULT '',
  PRIMARY KEY(`ID`),
  CONSTRAINT `FK_AgreementTemplateReference` FOREIGN KEY
`FK_AgreementTemplateReference` (`AgreementTemplateReference`)
    REFERENCES `agreementtemplate` (`ID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT
)
ENGINE = InnoDB;
```

- As a “root” (administrator of the database) account, create user “catnets” with password “catnets” that has all privileges on the database CAP_AgreementTemplates_DB. Or use the MySQL Adminitrator module to set up the privileges for user “catnets” to `CAP_AgreementTemplates_DB` database.

```
mysql> GRANT all privileges on CAP_AgreementTemplates_DB.* to
'catnets'@'localhost' identified by 'catnets';
mysql> GRANT all privileges on CAP_AgreementTemplates_DB.* to 'catnets'@'%'
identified by 'catnets';
```

5. MGS Agreements Repository

MGS Agreements Repository hosts the agreement templates received and agreement offers send by the MGS.

- Create a database named MGS_Agreement_DB.

```
mysql> CREATE DATABASE MGS_Agreement_DB;
```

- Create a tabled named MGS_Agreements within the MGS_Agreement_DB.

```
mysql> use MGS_Agreement_DB
mysql> CREATE TABLE `mgs_agreement_db`.`MGS_Agreements` (
  `ID` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  `AgreementTemplateReceived` TEXT NOT NULL DEFAULT '',
  `AgreementOfferSend` TEXT NOT NULL DEFAULT '',
  `Decision` TEXT NOT NULL DEFAULT '',
  PRIMARY KEY(`ID`)
)
ENGINE = InnoDB;
```

- As a “root” (administrator of the database) account, create user “catnets” with password “catnets” that has all privileges on the database MGS_Agreement_DB. Or use the MySQL Adminitrator module to set up the privileges for user “catnets” to `MGS_Agreement_DB` database.

```
mysql> GRANT all privileges on MGS_Agreement_DB.* to 'catnets'@'localhost'
identified by 'catnets';
```

```
mysql> GRANT all privileges on MGS_Agreement_DB.* to 'catnets'@'%'
identified by 'catnets';
```

B.2 Cat-DataMining prototype

1. Users' Loader Jobs Repository

Commands to filling the table with data:

```
mysql> INSERT INTO loader_db.userloaderjobs VALUES (NULL,'cardiff','cat-
J48DataMining_WS','http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.arff'
);

mysql> INSERT INTO loader_db.userloaderjobs VALUES (NULL,'cardiff','cat-
converterCsv2Arff_WS-
J48DataMining_WS','http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.csv')
;
```

Check that entry data is correct:

```
mysql> SELECT * FROM loader_db.userloaderjobs;
```

2. Catallactic Access Point Repository

Catallactic Access Point Repository hosts the agreement templates and agreement offers sent by the MGS.

- Fill in the table `agreementtemplate` from `cap_agreementtemplates_db` database with the following data:

```
mysql> INSERT INTO cap_agreementtemplates_db.agreementtemplate VALUES
(NULL,'cat-J48DataMining_WS','J48Service','AT-
J48Service','<AgreementTemplateLite><Name>J48-
ComplexService</Name><Context><AgreementInitiator></AgreementInitiator><Sta
rtingTime></StartingTime><TerminationTime></TerminationTime></Context><Term
s><BasicServiceType>J48</BasicServiceType><NumberOfBasicServiceNodes><!--
between 1 to 10 --
></NumberOfBasicServiceNodes><BasicServiceConstraints><ResponseTimePerReque
st>10<!-- maximum milliseconds --
></ResponseTimePerRequest></BasicServiceConstraints><PayForService></PayFor
Service></Terms></AgreementTemplateLite>');


mysql> INSERT INTO cap_agreementtemplates_db.agreementtemplate VALUES(NULL,
'cat-converterCsv2Arff_WS-J48DataMining_WS','ConverterCSVToArff-
J48Service','AT-ConverterCSVToArff-
J48Service','<AgreementTemplateLite><Name>ConverterCSVToARFF-J48-
ComplexService</Name><Context><AgreementInitiator></AgreementInitiator><Sta
rtingTime></StartingTime><TerminationTime></TerminationTime></Context><Term
s><BasicServiceType>ConverterCSVToARFF</BasicServiceType><BasicServiceType>
J48</BasicServiceType><NumberOfBasicServiceNodes><!-- between 1 to 10 --
></NumberOfBasicServiceNodes><BasicServiceConstraints><ResponseTimePerReque
st>10<!-- maximum milliseconds --
></ResponseTimePerRequest></BasicServiceConstraints><PayForService></PayFor
Service></Terms></AgreementTemplateLite>');
```

MySQL SCRIPT(already done at point A.4): The following script performs all operations needed: cap-agreementtemplates-script.sql

3. Converter .csv format to .arff format

(This is used for the Triana workflow – see section 7)

3.1 User Loader Job Repository

```
mysql> INSERT INTO loader_db.userloaderjobs VALUES
(NULL,'cardiff','converterCsv2Arff_WS','http://users.cs.cf.ac.uk/L.Joita/da
taFiles/weather.csv');
```

a. CAP Repository

```
mysql> INSERT INTO cap_agreementtemplates_db.agreementtemplate VALUES
(NULL, 'converterCsv2Arff','Converter_CsvToArff_Service','AT-
Converter_CsvToArff_Service',
'<AgreementTemplateLite><Name>Converter_CsvToArff-
ComplexService</Name><Context><AgreementInitiator></AgreementInitiator><Sta
rtingTime></StartingTime><TerminationTime></TerminationTime></Context><Term
s><BasicServiceType>Converter_CsvToArff</BasicServiceType><NumberOfBasicSer
viceNodes><!-- between 1 to 10 --
></NumberOfBasicServiceNodes><BasicServiceConstraints><ResponseTimePerReque
st>10<!-- maximum milliseconds --
></ResponseTimePerRequest></BasicServiceConstraints><PayForService></PayFor
Service></Terms></AgreementTemplateLite>');
```

B.3 Metrics for the Prototype

1. Repository for holding the metrics measured at the prototype level

Create a repository "`metrics_prototype`" that holds the metrics measured at the prototype level:

```
mysql> CREATE DATABASE metrics_prototype;
```

- Create a tabled named `metrics` within the `metrics_prototype`.

```
mysql> use metrics_prototype
mysql> CREATE TABLE `metrics_prototype`.`metrics` (
  `ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `ServiceProvisionTime` BIGINT UNSIGNED COMMENT 'Represents the time
required for MGS to complete a request for a service. Measured in
milliseconds',
  `ServiceExecutionTime` BIGINT UNSIGNED COMMENT 'Represents the job
execution time. Measured in milliseconds',
  `SuccessfulAllocationRate` DOUBLE UNSIGNED COMMENT 'The rate of
successful service allocation.',
  `UnsuccessfulAllocationRate` DOUBLE UNSIGNED COMMENT 'The rate of
unsuccessful service allocation.',
  PRIMARY KEY(`ID`)
)
ENGINE = InnoDB;
```

- As a “root” (administrator of the database) account, create user “catnets” with password “catnets” that has all privileges on the database MGS_Agreements_DB. Or use the MySQL Adminitrator module to set up the privileges for user “catnets” to `metrics_prototype` database.

```
mysql> GRANT all privileges on metrics_prototype.* to 'catnets'@'localhost'
identified by 'catnets';
```
mysql> GRANT all privileges on metrics_prototype.* to 'catnets'@'%' identified by 'catnets';

Annex C – Triana Integration

C.1 Cat-COVITE prototype and Triana

CAP getAgreementTemplate() invoked – ask for agreement template of the Query complex service

```
*************local request No:0 target:*************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
       uuid:98817970-166d-11db-8ad0-80cd5c234e90
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
       http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
         http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:getAgreementTemplate
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <serviceName xsi:type="xsd:string">
         cat-covite
        </serviceName>
      </ns1:getAgreementTemplate>
    </soapenv:Body>
  </soapenv:Envelope>
```

CAP getAgreementTemplate() invoked – send the agreement template of Query Complex Service

```
*************remote response No:0 target:*************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:getAgreementTemplateResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
&lt;AgreementTemplateLite&gt;&lt;Name&gt;QueryComplexService&lt;/Name&gt;&lt;Context
&gt;&lt;AgreementInitiator&gt;&lt;/AgreementInitiator&gt;&lt;StartingTime&gt;&lt;/StartingTi
me&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;&lt;/Context&gt;&lt;Terms&gt;&lt;B
asicServiceType&gt;QueryService&lt;/BasicServiceType&gt;&lt;NumberOfBasicServiceNodes&
gt;1 to 10&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;DBType&gt;Archi
tectural/Engineering/Construction&lt;/DBType&gt;&lt;ResponseTimePerRequest&gt;10&lt;/Res
ponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;&lt;/PayForS
ervice&gt;&lt;/Terms&gt;&lt;/AgreementTemplateLite&gt;
        </ns1:getAgreementTemplateReturn>
      </ns1:getAgreementTemplateResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.1 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

Full details of the SOAP messages exchanged by services within the Cat-COVITE prototype integrated into Triana workflow are shown in Listing C.1 -- the SOAP message request from MGS to the CAP Web Service, by invoking the getAgreementTemplate() method. The address of the Web Service invoked is highlighted in blue, as well as the request passed from MGS to CAP (the "cat-covite"). The CAP identifies that the request is for a Query Complex Service, therefore an agreement template of this complex service is invoked and passed back to the request initiator (MGS).

CAP_receiveAgreementOffer() invoked – An agreement offer is sent from MGS to CAP

```
**************local request No:1 target:**************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
       uuid:9b15dff0-166d-11db-8ad0-80cd5c234e90
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
       http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
         http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:receiveAgreementOffer
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <agreementOffer xsi:type="xsd:string">
&lt;AgreementOfferLite&gt;&lt;Name&gt;QueryComplexService&lt;/Name&gt;&lt;Context&gt;
&lt;AgreementInitiator&gt;cardiff&lt;/AgreementInitiator&gt;&lt;StartingTime&gt;&lt;/Startin
gTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;&lt;/Context&gt;&lt;Terms&gt;&l
t;BasicServiceType&gt;QueryService&lt;/BasicServiceType&gt;&lt;NumberOfBasicServiceNod
es&gt;1 to 10&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;DBType&gt;Arch
itectural/Engineering/Construction&lt;/DBType&gt;&lt;ResponseTimePerRequest&gt;10&lt;/R
esponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;100&lt;/P
ayForService&gt;&lt;/Terms&gt;&lt;/AgreementOfferLite&gt;
        </agreementOffer>
        <applInstance xsi:type="xsd:string">
         cat-covite
        </applInstance>
        <senderName xsi:type="xsd:string">
         cardiff
        </senderName>
      </ns1:receiveAgreementOffer>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.2 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

Listing C.2 details the SOAP message exchanged while the agreement offer has been filled in and the outcome is sent back to the CAP. The highlighted string represents the agreement offer made by the MGS in order to get the corresponding complex service (in this case, the Query Service) from the Catallactic GMM.

CAP_receiveAgreementOffer() – a decision is returned by CAP – either an EPR or nothing

```
**************remote response No:1 target:**************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:receiveAgreementOfferResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:receiveAgreementOfferReturn xsi:type="xsd:string">
          http://131.251.128.7:18088/axis/QueryServiceJWS/QueryService.jws
        </ns1:receiveAgreementOfferReturn>
      </ns1:receiveAgreementOfferResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

MGS_invokeService() – Cat-COVITE prototype instance requests job execution

```
**************local request No:2 target:**************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
        uuid:a784dbb0-166d-11db-8ad0-80cd5c234e90
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
        http://131.251.128.7:18088/axis/MGS23_QueryService.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:invokeQueryService
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <serviceURI xsi:type="xsd:string">
          http://131.251.128.7:18088/axis/QueryServiceJWS/QueryService.jws
        </serviceURI>
        <job xsi:type="xsd:string">
          SELECT * FROM SupplierDB.Product ORDER BY Price DESC;
        </job>
      </ns1:invokeQueryService>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.3 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

Listing C.3 shows the SOAP messages exchanged in which a service URI reference (end point reference) is returned or nothing. In the current version of this prototype, a static end point reference is always returned. The prototype makes use of this service URI and passes to it the job to be executed.

MGS_invokeService() – Cat-COVITE prototype instance returns the result

```
*************remote response No:2 target:*************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:invokeQueryServiceResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:invokeQueryServiceReturn xsi:type="xsd:string">
         &lt;?xml version=&quot;1.0&quot;?&gt;
&lt;dataset&gt;
7
manufacturer6
chair7
40
6
manufacturer5
chair6
35
5
manufacturer4
chair5
30
4
manufacturer4
chair4
25
3
manufacturer3
chair3
20
2
manufacturer2
chair2
15
1
manufacturer1
chair1
10
&lt;/dataset&gt;

        </ns1:invokeQueryServiceReturn>

      </ns1:invokeQueryServiceResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.4 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

Listing C.4 details the SOAP message received by the prototype, via the MGS, after invocation of the service URI received and passing the application job to be executed – the result of the job execution is highlighted.

C.2 Cat-DataMining prototype and Triana

C.2.1 Reference to the use case described in section 3.1

CAP getAgreementTemplate() invoked – ask for agreement template of the J48 Complex service

```
*************local request No:0 target:*************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
        uuid:b8deb970-165f-11db-ba78-aab8764a7b01
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
        http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:getAgreementTemplate
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <serviceName xsi:type="xsd:string">
          cat-J48DataMining_WS
        </serviceName>
      </ns1:getAgreementTemplate>
    </soapenv:Body>
  </soapenv:Envelope>
```

CAP getAgreementTemplate() invoked – send the agreement template of J48 Complex Service

```
*************remote response No:0 target:*************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:getAgreementTemplateResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
          &lt;AgreementTemplateLite&gt;&lt;Name&gt;J48-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;&lt;/AgreementInitia
tor&gt;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTi
me&gt;&lt;/Context&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;J48&lt;/BasicServiceType&gt;&
lt;NumberOfBasicServiceNodes&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimePe
rRequest&gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;
&lt;/PayForService&gt;&lt;/Terms&gt;&lt;/AgreementTemplateLite&gt;
        </ns1:getAgreementTemplateReturn>
      </ns1:getAgreementTemplateResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.5 - SOAP messages exchanged by MGS and Catallactic middleware (GMM)

CAP_receiveAgreementOffer() invoked – An agreement offer is sent from MGS to CAP

```
**************local request No:1 target:**************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
       uuid:064813a0-1660-11db-ba78-aab8764a7b01
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
       http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
         http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:receiveAgreementOffer
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <agreementOffer xsi:type="xsd:string">
         &lt;AgreementOfferLite&gt;&lt;Name&gt;J48-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;cardiff&lt;/Agreeme
ntInitiator&gt;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/Termin
ationTime&gt;&lt;/Context&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;J48&lt;/BasicServiceTy
pe&gt;&lt;NumberOfBasicServiceNodes&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimeP
erRequest&gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;
120&lt;/PayForService&gt;&lt;/Terms&gt;&lt;/AgreementOfferLite&gt;
        </agreementOffer>
        <applInstance xsi:type="xsd:string">
         cat-J48DataMining_WS
        </applInstance>
        <senderName xsi:type="xsd:string">
         cardiff
        </senderName>
      </ns1:receiveAgreementOffer>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.6 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

CAP_receiveAgreementOffer() – a decision is returned by CAP – either an EPR or nothing

```
**************remote response No:1 target:**************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:receiveAgreementOfferResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:receiveAgreementOfferReturn xsi:type="xsd:string">
          http://131.251.128.7:18088/axis/services/J48Classifier
        </ns1:receiveAgreementOfferReturn>
      </ns1:receiveAgreementOfferResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

MGS_invokeService() – Cat-DataMining J48 prototype instance requests job execution

```
**************local request No:2 target:**************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
        uuid:101b55e0-1660-11db-ba78-aab8764a7b01
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
        http://131.251.128.7:18088/axis/MGS21_J48_WS.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:invokeService
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <serviceURI xsi:type="xsd:string">
          http://131.251.128.7:18088/axis/services/J48Classifier
        </serviceURI>
        <job xsi:type="xsd:string">
          http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.arff
        </job>
      </ns1:invokeService>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.7 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

MGS_invokeService() – Cat-DataMining J48 prototype instance returns the result

```
**************remote response No:2 target:**************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:invokeServiceResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:invokeServiceReturn xsi:type="xsd:string">
         J48 pruned tree
------------------

play = no
|   temperature &lt;= 71: rainy (2.0)
|   temperature &gt; 71: sunny (3.0)
play = yes
|   humidity &lt;= 70: sunny (3.0/1.0)
|   humidity &gt; 70
|   |   temperature &lt;= 71: rainy (2.0)
|   |   temperature &gt; 71: overcast (4.0/1.0)

Number of Leaves  :     5

Size of the tree :      9

        </ns1:invokeServiceReturn>
      </ns1:invokeServiceResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.8 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

C.2.2 Reference to the use case described in section 3.2 – Case 1

CAP_getAgreementTemplate() invoked – ask for agreement template of Converter complex service

```
**************local request No:0 target:**************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
       uuid:2a6a3d50-1672-11db-b4a7-82ee7c1d70a1
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
       http://131.251.128.7:18088/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:getAgreementTemplate
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <serviceName xsi:type="xsd:string">
          converterCsv2Arff
        </serviceName>
      </ns1:getAgreementTemplate>
    </soapenv:Body>
  </soapenv:Envelope>

**************local request No:1 target:**************
```

CAP_getAgreementTemplate() invoked – ask for agreement template of J48 complex service

```
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
       uuid:2a67cc50-1672-11db-b4a7-82ee7c1d70a1
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
       http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:getAgreementTemplate
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <serviceName xsi:type="xsd:string">
          cat-J48DataMining_WS
        </serviceName>
      </ns1:getAgreementTemplate>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.9 - SOAP messages exchanged by MGS and Catallactic middleware (GMM)

CAP_getAgreementTemplate() invoked – send the agreement template of Converter complex service

```
**************remote response No:1 target:**************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:getAgreementTemplateResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
         &lt;AgreementTemplateLite&gt;&lt;Name&gt;Converter_CsvToArff-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;&lt;/AgreementInitiator&gt
;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;&lt;/C
ontext&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;Converter_CsvToArff&lt;/BasicServiceType&gt;&lt
;NumberOfBasicServiceNodes&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimePerRequ
est&gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;&lt;/P
ayForService&gt;&lt;/Terms&gt;&lt;/AgreementTemplateLite&gt;
        </ns1:getAgreementTemplateReturn>
      </ns1:getAgreementTemplateResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

CAP_getAgreementTemplate() invoked – send the agreement template of J48 complex service

```
**************remote response No:0 target:**************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:getAgreementTemplateResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:getAgreementTemplateReturn xsi:type="xsd:string">
         &lt;AgreementTemplateLite&gt;&lt;Name&gt;J48-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;&lt;/AgreementInitiator&gt
;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;&lt;/C
ontext&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;J48&lt;/BasicServiceType&gt;&lt;NumberOfBasicS
erviceNodes&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimePerRequ
est&gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;&lt;/P
ayForService&gt;&lt;/Terms&gt;&lt;/AgreementTemplateLite&gt;
        </ns1:getAgreementTemplateReturn>
      </ns1:getAgreementTemplateResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.10 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

CAP_receiveAgreementOffer() invoked–Agreement offer sent from MGS to CAP -- Converter Service

```
************local request No:2 target:************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
       uuid:48b11810-1672-11db-b4a7-82ee7c1d70a1
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
       http://131.251.128.7:18088/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:receiveAgreementOffer
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <agreementOffer xsi:type="xsd:string">
          &lt;AgreementOfferLite&gt;&lt;Name&gt;Converter_CsvToArff-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;cardiff&lt;/AgreementInitia
tor&gt;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;
&lt;/Context&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;Converter_CsvToArff&lt;/BasicServiceType&
gt;&lt;NumberOfBasicServiceNodes&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimePerRequ
est&gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;80&lt;/
PayForService&gt;&lt;/Terms&gt;&lt;/AgreemenOfferLite&gt;
        </agreementOffer>
        <applInstance xsi:type="xsd:string">
          converterCsv2Arff
        </applInstance>
        <senderName xsi:type="xsd:string">
          cardiff
        </senderName>
      </ns1:receiveAgreementOffer>
    </soapenv:Body>
  </soapenv:Envelope>
```

CAP_receiveAgreementOffer() –decision returned by CAP–either EPR or nothing – Converter Service

```
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:receiveAgreementOfferResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:receiveAgreementOfferReturn xsi:type="xsd:string">
          http://131.251.128.7:18088/axis/converters/Converter_CsvToArff_WS.jws
        </ns1:receiveAgreementOfferReturn>
      </ns1:receiveAgreementOfferResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.11 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

CAP_receiveAgreementOffer() –decision returned by CAP–either EPR or nothing – J48 Service

```
**************local request No:3 target:**************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
       uuid:5d18f0c0-1672-11db-b4a7-82ee7c1d70a1
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
       http://131.251.47.197:8080/axis/CatallacticAccessPoint/CatallacticAccessPoint.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
         http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:receiveAgreementOffer
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <agreementOffer xsi:type="xsd:string">
         &lt;AgreementOfferLite&gt;&lt;Name&gt;J48-
ComplexService&lt;/Name&gt;&lt;Context&gt;&lt;AgreementInitiator&gt;cardiff&lt;/AgreementInitiat
or&gt;&lt;StartingTime&gt;&lt;/StartingTime&gt;&lt;TerminationTime&gt;&lt;/TerminationTime&gt;&l
t;/Context&gt;&lt;Terms&gt;&lt;BasicServiceType&gt;J48&lt;/BasicServiceType&gt;&lt;NumberOfBas
icServiceNodes&gt;&lt;!-- between 1 to 10 --
&gt;&lt;/NumberOfBasicServiceNodes&gt;&lt;BasicServiceConstraints&gt;&lt;ResponseTimePerReque
st&gt;10&lt;!-- maximum milliseconds --
&gt;&lt;/ResponseTimePerRequest&gt;&lt;/BasicServiceConstraints&gt;&lt;PayForService&gt;70&lt;/
PayForService&gt;&lt;/Terms&gt;&lt;/AgreementOfferLite&gt;
        </agreementOffer>
        <applInstance xsi:type="xsd:string">
         cat-J48DataMining_WS
        </applInstance>
        <senderName xsi:type="xsd:string">
         cardiff
        </senderName>
      </ns1:receiveAgreementOffer>
    </soapenv:Body>
  </soapenv:Envelope>
```

CAP_receiveAgreementOffer() –decision returned by CAP–either EPR or nothing – J48 Service

```
                    remote response No:3 target:
  <?xml version="1.0" encoding="utf-8"?>
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
        <ns1:receiveAgreementOfferResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
          <ns1:receiveAgreementOfferReturn xsi:type="xsd:string">
           http://131.251.128.7:18088/axis/services/J48Classifier
          </ns1:receiveAgreementOfferReturn>
       </ns1:receiveAgreementOfferResponse>
      </soapenv:Body>
    </soapenv:Envelope>
```

Listing C.12 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

MGS_invokeService() – Cat-DataMining prototype instance (Converter & J48) returns the result

```
*************local request No:4 target:*************
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
      <wsa:MessageID soapenv:mustUnderstand="0">
        uuid:64b4b990-1672-11db-b4a7-82ee7c1d70a1
      </wsa:MessageID>
      <wsa:To soapenv:mustUnderstand="0">
        http://131.251.128.7:18088/axis/MGS22_Converter_J48_WS.jws
      </wsa:To>
      <wsa:From soapenv:mustUnderstand="0">
        <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
        </wsa:Address>
      </wsa:From>
    </soapenv:Header>
    <soapenv:Body>
      <ns1:invokeService soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <serviceURI_Converter xsi:type="xsd:string">
          http://131.251.128.7:18088/axis/converters/Converter_CsvToArff_WS.jws
        </serviceURI_Converter>
        <serviceURI_J48 xsi:type="xsd:string">
          http://131.251.128.7:18088/axis/services/J48Classifier
        </serviceURI_J48>
        <job xsi:type="xsd:string">
          http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.csv
        </job>
      </ns1:invokeService>
    </soapenv:Body>
  </soapenv:Envelope>

*************remote response No:4 target:*************
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:invokeServiceResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://DefaultNamespace">
        <ns1:invokeServiceReturn xsi:type="xsd:string">
          J48 pruned tree
------------------
play = no
|   temperature &lt;= 71: rainy (2.0)
|   temperature &gt; 71: sunny (3.0)
play = yes
|   humidity &lt;= 70: sunny (3.0/1.0)
|   humidity &gt; 70
|   |   temperature &lt;= 71: rainy (2.0)
|   |   temperature &gt; 71: overcast (4.0/1.0)
Number of Leaves  :     5
Size of the tree :      9
        </ns1:invokeServiceReturn>
      </ns1:invokeServiceResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Listing C.13 – SOAP messages exchanged by MGS and Catallactic middleware (GMM)

This document first gives an introduction to Application Layer Networks and subsequently presents the catallactic resource allocation model and its integration into the middleware architecture of the developed prototype. Furthermore use cases for employed service models in such scenarios are presented as general application scenarios as well as two very detailed cases: Query services and Data Mining services. This work concludes by describing the middleware implementation and evaluation as well as future work in this area.

ISSN 1864-9300