Bayreuther Arbeitspapiere zur Wirtschaftsinformatik

Oscar Ardaiz, Pablo Chacin, Isaac Chao, Felix Freitag, LeandroNavarro (Universidad Polytecnica de Catalunya), Liviu Joita, Omer F. Rana (Cardiff University), Werner Streitberger, Sebastian Hudert, Torsten Eymann (University of Bayreuth)

Proof-of-Concept Application - Annual Report Year 3

Bayreuth Reports on Information Systems Management

UNIVERSITÄT BAYREUTH

Authors:

Oscar Ardaiz, Pablo Chacin, Isaac Chao, Felix Freitag, LeandroNavarro (Universidad Polytecnica de Catalunya), Liviu Joita, Omer F. Rana (Cardiff University), Werner Streitberger, Sebastian Hudert, Torsten Eymann (University of Bayreuth)

IST-FP6-003769 CATNETS

D3.3

Annual Report on WP3 (T0+36)

Contractual Date of Delivery to the CEC:	31 August 2007
Actual Date of Delivery to the CEC:	30 September 2007
Author(s):	Oscar Ardaiz, Pablo Chacin, Isaac Chao, Felix Freitag, Liviu Joita, Leandro Navarro, Omer F. Rana, Werner Streitberger
Work package:	WP3
Est. person months:	9.5
Security:	public
Nature:	final
Version:	1.0
Total number of pages:	53

Abstract:

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

(1) Development of performance measuring components for experiments. The task is mainly concern to defining the necessary components for the generation of the metrics, as well as capturing these values;

(2) Development of a distributed application to execute on economic-enhanced Grid/P2P platform and middleware integration. The task is related to the development of the prototype, including architecture, layers, modules, services, deployment, tests and capture of metrics, which are further analysed in WP4.

Keyword list: Prototype, Grid Market Middleware, Resource Allocation

CATNETS Consortium

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

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: Christof Weinhardt
E-mail: weinhardt@iism.uni-karlsruhe.de

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

University of Mannheim
Chair of Business Administration and Information Systems
- E-Business and E-Government -
L9, 1-2
68131 Mannheim
Germany
Tel: +49 621 181 3321, Fax: +49 621 181 3310
Contactperson: Daniel Veit
E-mail: veit@uni-mannheim.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

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

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
01	16/05/2007	LJ	First draft, Table of Content
02	23/06/2007	LJ	Chapters 1, 3, 4 - first draft
03	06/07/2007	IC	Chapter 2, 4 - first draft
04	13/07/2007	IC	More content added to chapters 2 and 4
05	18/07/2007	SH	Content added to chapters 2 and 4
06	27/07/2007	LJ	Content added to chapter 7, revision to chapters 1, 3, and 4
07	30/07/2007	LJ	Chapter 5 and 6 added
08	07/08/2007	LJ	Revision to chapters 1, 3, 4, 5, 6 and 7
09	20/08/2007	IC	More content added to chapters 2 and 4
10	23/08/2007	IC	Revision of chapters 2, 3, 4, 5
11	28/08/2007	LJ	Revision, proof reading
12	28/09/2007	WS	Final draft editing
13	28/09/2007	LJ	Revision chapter 4
14	01/10/2007	IC	Revision algorithm ZIP agents sec. 2.3.2 + alignment WP4
15	04/10/2007	WS	Incorporating revisions of chapter 6

TABLE OF CONTENT:

1 Introduction

This deliverable describes the work undertaken in Work Package 3 (WP3) of the CATNETS project, which involves the following tasks:

- T3.3 Performance measuring components for experiments. The task is mainly concern to defining the necessary components for the generation of the metrics, as well as capturing these values.

- T3.4 Distributed application to execute on economic-enhanced Grid/P2P platform and middleware integration. The task is related to the development of the prototype, including architecture, layers, modules, services, deployment, tests and capture of metrics, which are further analysed in WP4.

The document is divided in seven parts:

- The introduction into the Application Layer Network;

- Catallactic resource allocation model and middleware architecture;

- Distributed application to execute on economic-enhanced Grid/P2P platform and middleware integration;

- Prototype performance measuring components for experiments;

- Relation to other work packages;

- A case study of the CATNETS applicability for an experimental National Visualization Service in UK, and conclusions.

2 Catallactic Middleware

The Grid Market Middleware (GMM) design, implementation and early evaluation have been thoroughly described in previous deliverables. Development in core GMM components (agent hosting, group communication, metric collection) has been frozen since Year 2, once requirements have been met. For this last year, effort has been switched to two main tasks: firstly, specifying and implementing a dedicated resource model amenable to detailed experimentation with the prototype in a resource constrained environment; secondly, testing as many economic protocols as possible. So far we have implemented three different economic algorithms in the same GMM substrate. We have demonstrated with this the modularity of the tool referred as Grid Market Middleware toolkit.

High-Performance Grid Computing (HPGC) combines the advances in research and technologies in high speed networks, software, distributed computing and parallel processing to deliver high-performance, large-scale and cost-effective computational, storage and communication capabilities to a wide range of applications. Typically the resources are used in Grids to provide processing capabilities for massive batch job scheduling. In this context, the resources are consumed in a dedicated mode. Each processor runs one batch job at a time, and concurrent processing of several jobs in one processor is not considered. This contrasts with transactional applications. With transactional applications there are different issues to be considered, such as: accessing databases or maintaining the transactional integrity of the request. Resources are not dedicated, and small concurrent computations for specific tasks might be performed continuously on the resources. We considered into the experimental phase of the prototype running in the cluster only the HPGC scenario with full dedicated resources.

2.1 Catallactic Resource Allocation Model

The dedicated resource model translates for the market concerns in a limited offer by the BasicServices (BSs). Once the BS has sold its resource to a ComplexService (CS), it cannot accept more bids from other CSs' Call for Proposals (CFPs) until the moment when the client of the awarded CS ends the execution of the sold service (i.e. the Data Mining service in our prototype scenario) in the resource. This peculiarity is what makes novel and interesting this scenario with respect to other previous research on decentralized economic algorithms based on simple learning agents in ideal markets [Preist, 1998], [Despotovic, 2004].

The dedicated resource model is modelled in this precise way for the GMM deployment in clusters:

1) Each BS is deployed in one node. The CPU of this node is its resource. Then the second market is useless here, we consider that the BS always has this precise resource and no other resource available.

2) Once a CS buys a BS, the CS (which could be seen, as a logical element, the application proxy which executes the service with the end point reference (EPR) passed by the BS) executes the service (i.e. Data Mining service) on the resource and no other CS can bargain for it. When execution is over, the resource can come back to the market. The BSs are aware of execution termination by means of a lock that the CS deletes on the resource when execution ends.

2.2 Catallactic Middleware Infrastructure

There has been an increase of interest in Service Oriented Grids (SOGs) recently within the Grid community towards services that are often considered as a natural progression from component based software development, and as a mean to integrate different component development frameworks. A service in this context may be defined as a behaviour that is provided by a component to be used by any other component. A service stresses interoperability and may be dynamically discovered and used [Foster, 2002]. Utility computing assumes service instances are created on the fly and automatically bound as applications that are configured dynamically. The service viewpoint abstracts the infrastructure level of an application. It enables the efficient usage of Grid resources and facilitates utility computing, especially when redundant services can be used to achieve fault tolerance. As for configurability, a SOG system is configured on-demand and flexibly, which means different components are bound to each other late in the composition process. Thus, the configuration can change dynamically as needed and without loss of correctness.

In a SOG infrastructure, the GMM is exposed to be accessed by applications trough a convenient access point, a Web Service which can be deployed in any application server and integrated as a service in an existing SOG. Figure 2.1 describes the main steps in the interaction trough the access point. When a client issues a request, the application determines which Grid services are required to fulfil it. These Grid services represent either software services (e.g. a data processing algorithm) or computational resources. The application translates these requirements into a standardized Service Level Agreement (SLA) document based on WS-Agreement [WS-Agreement, 2007]. The application invokes the access point and passes the corresponding SLA request. This is in turn parsed and processed at the access point, which instantiates the GMM with the required economic agents to fulfil the client request.

The GMM searches among the available service providers, which have registered their particular service specifications, like 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 on behalf of the applications as buyers. Once an agreement is reached between the trading agents, a Grid service instance is created for the application and a reference is returned to the application/client, which can invoke it.

The server-side infrastructure is deployed by a set of scripts which allow for the bootstrapping of BSs in available resources. The scripts perform the automatic deployment and configuration of the BSs, which are then ready to be contacted by CSs. Services offered by BSs for clients executions are also deployed and exposed in Apache Tomcat application servers. Complemented by the access point, this comprises a complete infrastructure for economic-based SOGs.
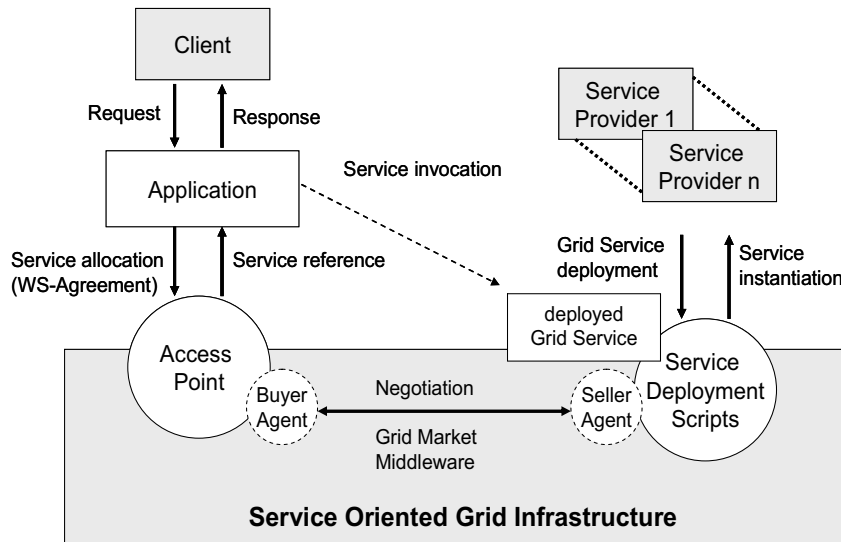
Figure 2.1 - Service Oriented Grid (SOG) infrastructure.

2.3 Grid Market Middleware and its implementation

The GMM allow currently for plugging of various decentralized economic algorithms. We have demonstrated this with three implementations: the Catallactic agents implementation discussed in WP1 and the alternative algorithms such as: Contract-Net protocol with simple offer/demand, and Zero Intelligence Plus (ZIP) agents in a continuous double auction.

2.3.1 Contract Net simple offer/demand agents

In this version of the GMM, we have plugged in as economic agents an implementation of the Contract-Net protocol standardized by FIPA [FIPA, 2007], as shown in Figure 2.2. The Contract-Net protocol has been developed to specify problem-solving communication and control for nodes in a distributed problem solver. Task distribution is affected by a negotiation process, a discussion carried on between nodes with tasks to be executed and nodes that may be able to execute those tasks. This protocol has been applied to many domains in multi-agent systems [Paurobally, 2004]. An example of application of the protocol to Grid resource allocation can be found in [Chao, 2004].
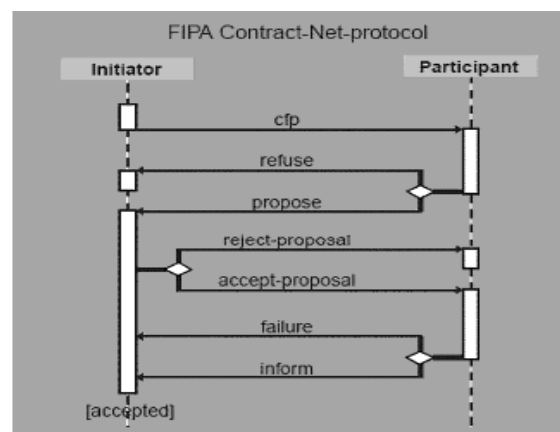
Figure 2.2 - Contract-Net [taken from FIPA web site]

The Contract-Net protocol starts with a task announcement phase by the initiator (the buyer), which can be answered by one or more participants (the sellers). This announcement is carried out by a groupcast of a call for proposals (CFP). After concluding this period, the initiator selects from the set of collected proposals the best one, informing the winner. On top of this protocol we apply a simple offer/demand-based economic algorithm: the sellers will answer the CPFs which meet its current selling price. If the CFP does not meet its requirements, the seller will lower its expectations and it will decrease the selling price. As for the buyers, if a seller rejects the CFP, then it will lower its expectation by increasing the offer in the next CFP. Both the buyers and the sellers will increase their expectations in case of receiving offers/bids which meet their expectations. The price updating is done at fixed small price steps.

2.3.2 Zero Intelligence Plus agents

In this version we consider a simplified Grid market with only one homogeneous Data Mining service being traded. The execution time of the service can be varied during the experiments. The auction mechanism is a continuous double auction in which the agents follow a modified ZIP strategy based on [Preist, 1998].

In the context of the GMM, the buyer agents are called ComplexServices (CSs) and the seller agents BasicServices (BSs). CSs aggregate BSs from the market. As BSs and CSs get involved in trading, the price will evolve by offer and demand, with dependence on the limited CS budget and the limited resources which can be sold by the BS. Once the BS has sold its resource to a CS, it cannot accept more bids from other CSs' CFPs until the moment when the client of the awarded CS ends the execution of the sold Data Mining service in the resource.

For the realization of the decentralized continuous double auction we divide the traders in subgroups, called clusters (see Figure 2.3) which are trading independently. This allows to cope with the scalability of large networks. Moreover this approach enables the agents to be well-informed of shouts from other agents, which in decentralized auctions is a general problem [Despotovic, 2004]. To avoid this groups are only trading isolated, agents have to join and leave the clusters. The selection of individual agents to move to another cluster depends on their trading success. This method allows reaching one global equilibrium price *P0* for all clusters situated the distributed market place. As the feasibility of a global *P0* is already shown in [Ogston, 2002], we will concentrate our prototype analysis focusing in one bidding cluster.
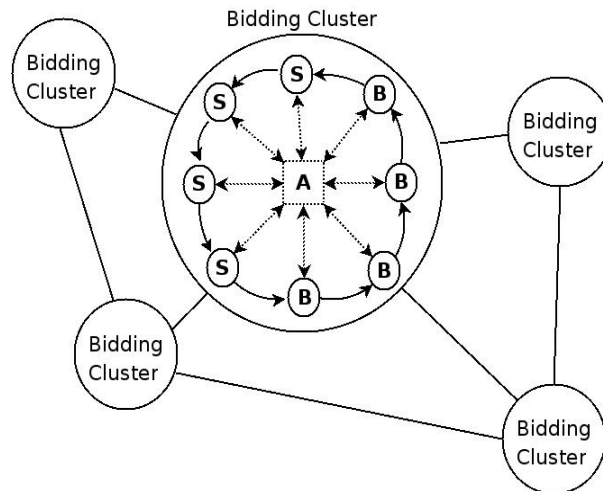
Figure 2.3 - Bidding clusters

Each cluster deploys an own central continuous double auction. The agents are coordinated in synchronous manner and are acting in bidding rounds. Therefore a delegated auctioneer controls the matching of the bids and offers, where the highest bid corresponds to the lowest offer. No matching of a trade will be executed, if no offers are lower than the highest bid exists.

The bidding algorithm is based on extended ZIP agents. This allows reaching the equilibrium price *P0*, at which the maximum resources will be exchanged, with simple agents. Therefore they have to know the minimum price of the shouted offers, by sellers *Smin* and the maximum price of the shouted bids by buyers *Bmax*. The agents base the bidding algorithm to calculate the new price *P(t+1)* on these two values. 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.

In the following algorithm, PriceChange is a parameter that defines the price change rate. The algorithm bases on a momentum γ considering the weight of previous price changes and on the learning rate β describing the rapidity of adoption to the current target price.

> **Algorithm1** : Bidding algorithm of the BS (seller).
> **Input:** random1 > 0 and < 0.2
> **Input:** random2 > 0 and < 0.2 and not random1;
> **if** Smin > Bmax **then**
>   PT = Smin - (random1 * P(t) + random2);
> **else**
>   PT = Bmax + (random1 * P(t) + random2);
> **endif**
> priceChange = γ * priceChange + (1-γ) * β * (PT-P(t));
> P(t+1) = maximum (P(T)+priceChange, Pmin) ;
>
> **Algorithm 2**: Bidding algorithm of the CS (buyer).
> **Input:** random1 > 0 and < 0.2
> **Input:** random2 > 0 and < 0.2 and not random1;
> **if** Smin > Bmax **then**
>   PT= Bmax + (random1 * P(t) + random2);

else
    PT = Smin - (random1 * P(t) + random2);
endif
priceChange = γ * priceChange + (1-γ) * β * (PT-P(t));
P(t+1) = minimum (P(T)+priceChange, budget) ;

2.3.3 Catallactic agents

The catallactic agents are more complex than both simple Contract-Net agents and ZIP agents from previous sections. In fact, catallactic agents apply and iterate Contract Net protocol with sophisticated strategies for the bargaining (the catallactic strategy described thoroughly in WP1) and applying an extra evolutionary learning step (conform to the learning algorithm from WP1). These sophistications target both a better control of agents' goals through the strategy, and improve adaptability through evolutionary learning.

Figure 2.4 depicts the whole process of the middleware implementation. The ApplicationProxyAgent firstly creates a demand message and sends this to all the ComplexServiceAgents (CSA) in the market (groupcast). After the discovery timeout occurred the ApplicationProxyAgent chooses one of the CSA that answered to that demand message for processing it on the actual CATNETS market (acknowledgment of the demand for the one chosen CSA). After this first step the CSA task to handle the particular demand is identified. This agent subsequently creates a Call-for-Proposals (CfP) message according to the demand received (considering the types and numbers of needed BasicServices for this ComplexService). After creating the CfP message, this is sent to the Service Market (groupcast to all BasicServicAgents). Again, the sending agent waits for a distinct amount of time, the discovery timeout, and then chooses the best offer of all received answers to this CfP (proposals from BasicServiceAgents). Subsequently, the CSA starts a one-to-one negotiation with the chosen BasicServiceAgent (BSA). After successfully finishing this negotiation the BSA creates a CfP message for the Resource Market according to the agreement just reached with the CSA. Again after reaching the discovery timeout for the Resource Market the BSA starts a one-on-one negotiation with the Resource Agent having posted the best proposal after the CfP.
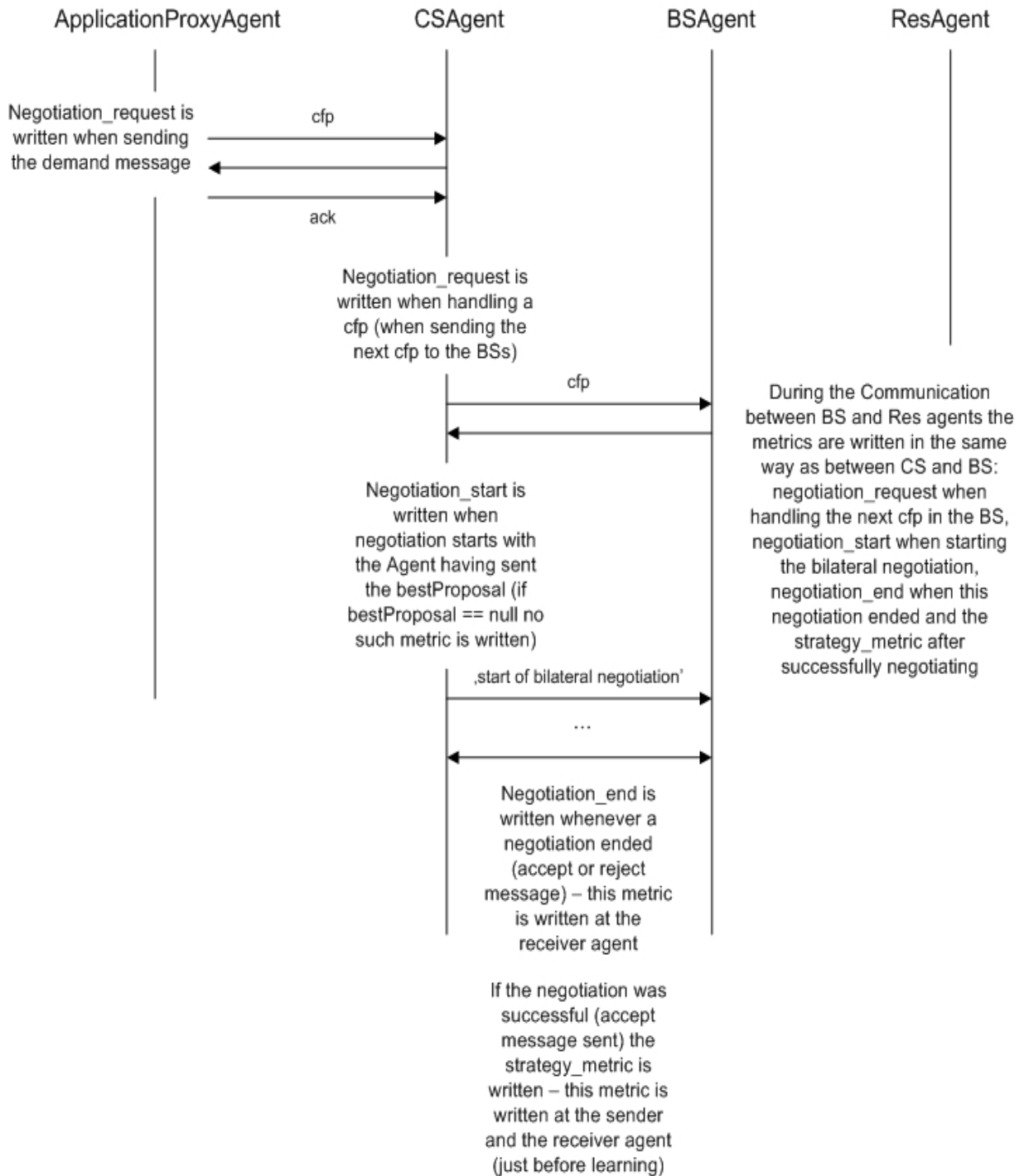
Figure 2.4 – Abstract sequence diagram of catallactic middleware agent process

3 Distributed application to execute on economic-enhanced Grid/P2P platform and middleware integration

Different examples of application scenarios can be constructed and could benefit from using the Grid Market Middleware (GMM) in combination with centralized auctions or decentralized bilateral bargaining approaches 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.

One application scenario could require 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 [Schnizler, 2006] 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.

Figure 3.1 details the logical components of a distributed catallactic enabled prototype along the three layers: the application layer, the catallactic middleware layer and the base platform layer – in the use case of one service invocation [WP3-Year2, 2006].
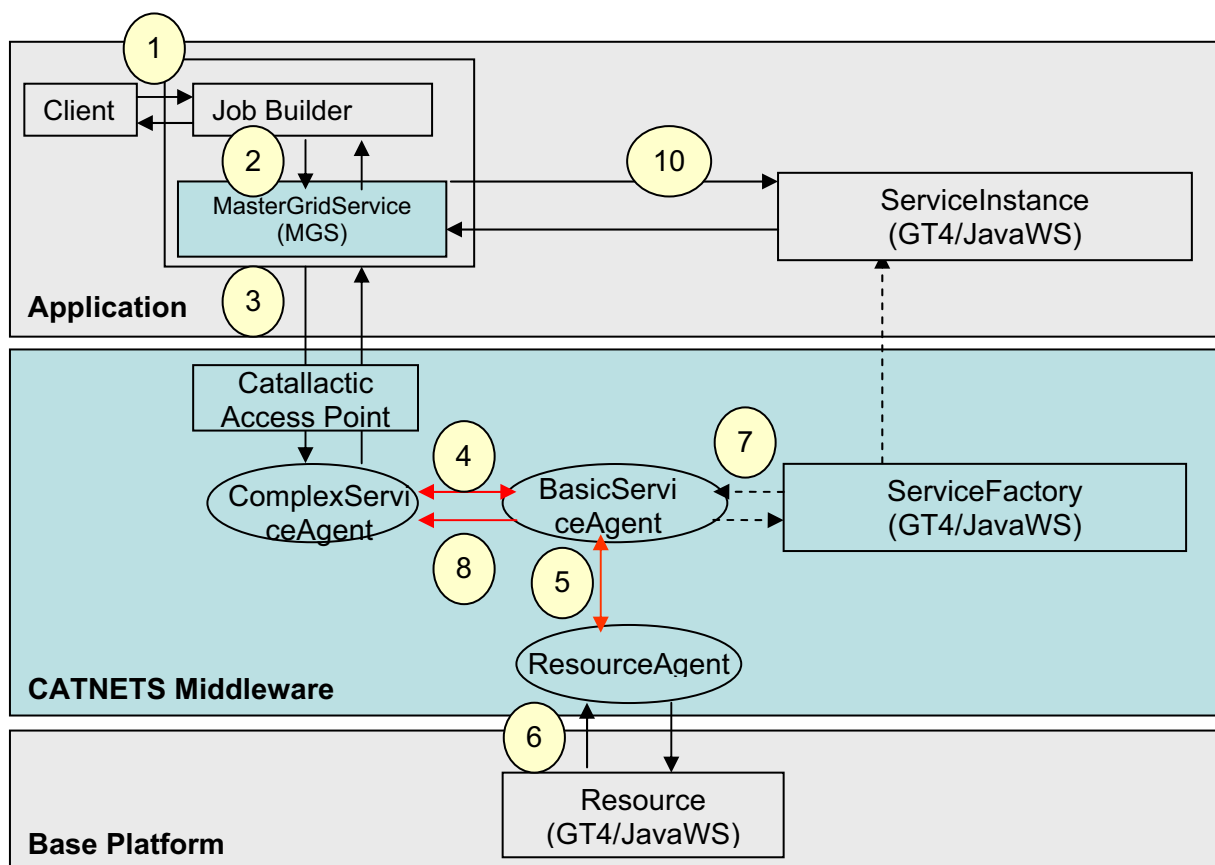
Figure 3.1 – Catallactic enabled prototype – logical layers and components

At the application layer, there are two main logical modules: (1) the job builder module, which interprets the user's request, and (2) the Master Grid Service (MGS), the module named in Catnets that requests services on behalf of the user. The application requests are made via an interface, named Catallactic Access Point (CAP), this is an important module that 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 basic 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.

The following sub-sections contain a description of two layer components of the proof-of-concept prototype: the Application layer and the Middleware layer, as well as the integration and deployment process as one entity distributed application.

3.1 Application Layer Components

Figure 3.2 shows a detailed graphical view of components and repositories of an application prototype that accesses 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. In our implementation, the repositories are based on MySQL 5.0. The Catallactic Access Point Web Service (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 designed, developed, implemented and deployed are presented in [WP3-Year2, 2006] -- these involve the use of a Query Service and a Data Mining Service.
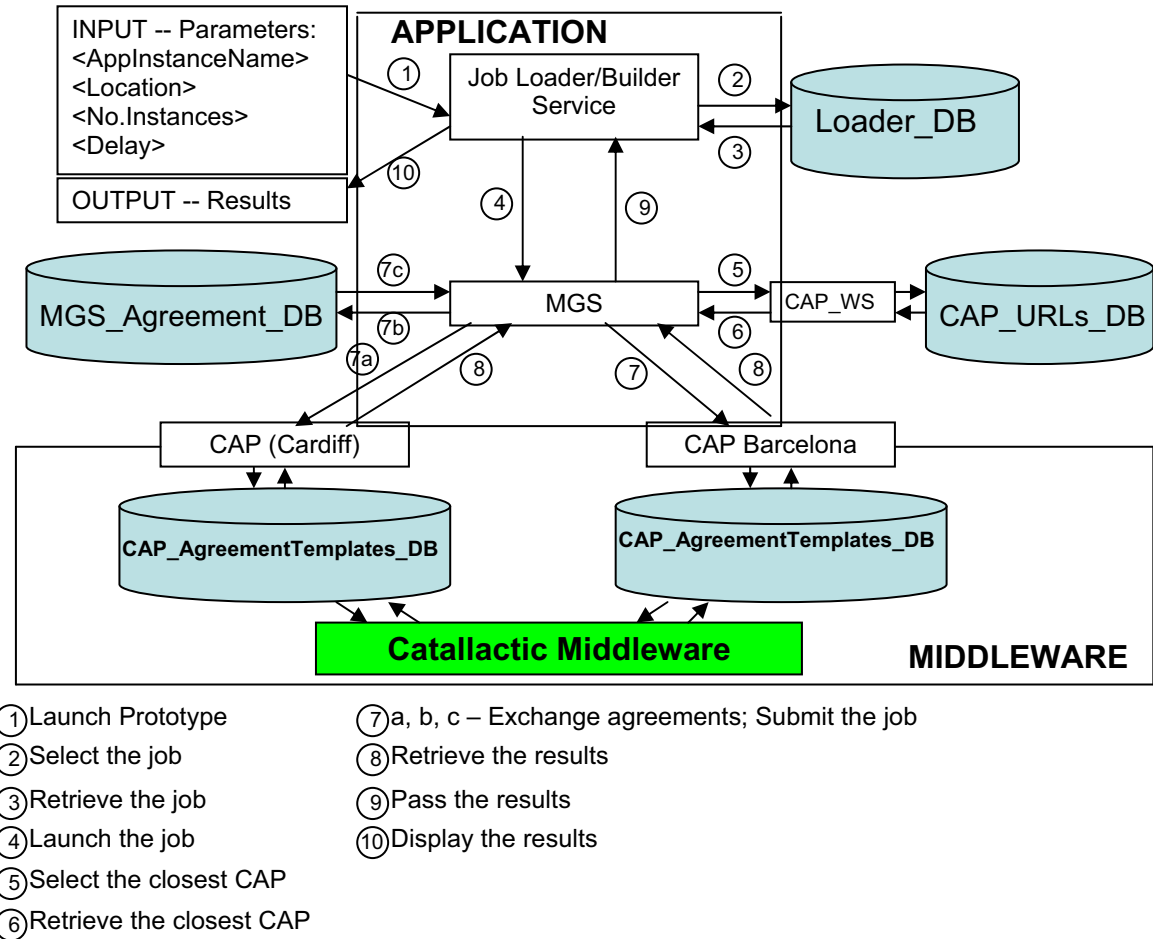
Figure 3.2 – Application prototype components

The workflow process, in a generalised application that connects to a Catallactic GMM, could be summarised as:

1 – A user enters the request criterion.

2 – A type of Job Loader / Builder component service interprets the user's request and converts it into a service request.

3 – A generic service, interpreted by Master Grid Service in the above diagram, 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.

3.1.1 Scenario Generator

The generation of a scenario at the prototype can be done manually, using a GUI which allows the user to specify the initial parameters the prototype needs to start with. Figure 3.3 presents the GUI. The GUI provide an interface for the prototype to set up the scenario that application will run for testing the capabilities of Catallaxy, as well as to perform tests and extract the results which will be compare with the simulator results in WP4.
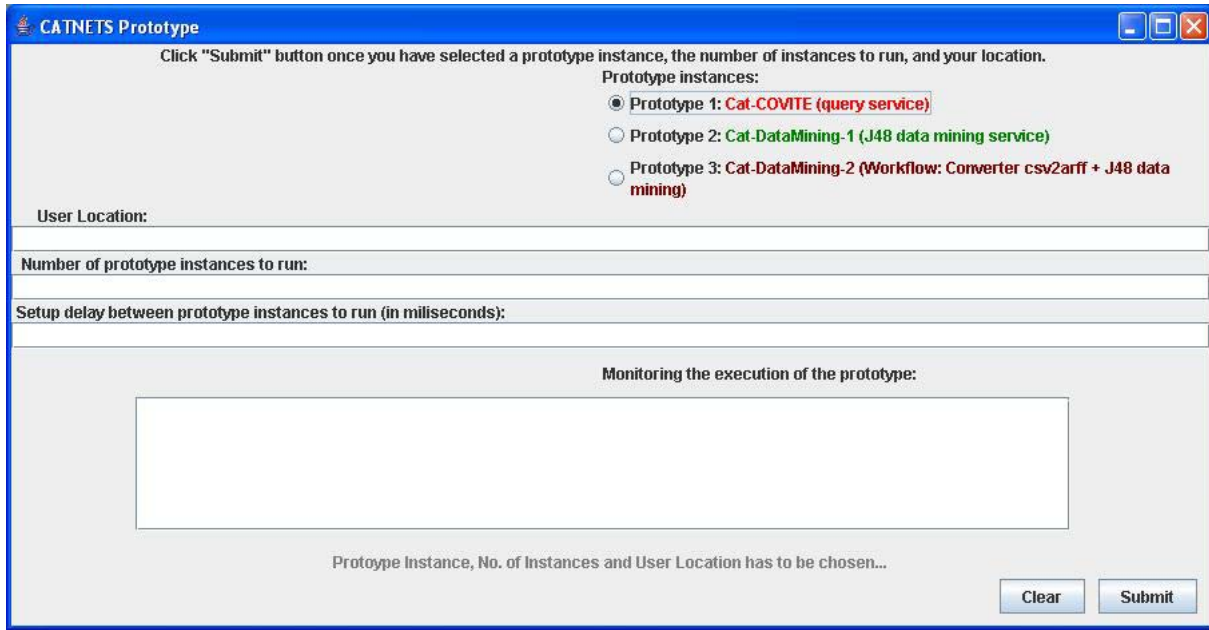
Figure 3.3 – CATNETS prototype

The use of the GUI is straightforward: the user specifies which application prototype instance to start by choosing among the buttons from the Prototype Instances layer. The implementation provides three different types of instances:

- firstly, Cat-COVITE prototype, which is based on the COVITE prototype [COVITE].
- secondly, the Cat-DataMining-1 prototype, in which one service request for and invocation of a J48 data mining service is made.
- thirdly, the Cat-DataMining-2 prototype, in which a workflow of services could be requested to GMM via the CAP. This option is available only in a centralized case, in which the services that compose the workflow are allocated statically by the CAP.

In order to run the tests, Figure 3.4 provides an example of the prototype GUI with parameters. The prototype instance to run is Cat-COVITE, the user location is: cardiff – this will be used to asses the closest CAP that the application will invoke in order to request the necessary services from the markets (GMM). Also, the market instance is connected to the CAP. The "number of prototype instances to run" and the "delay between prototype instances to run" are two parameters needed to run tests, collect the metrics obtained in the prototype and compare the results with the simulator that runs under similar conditions.
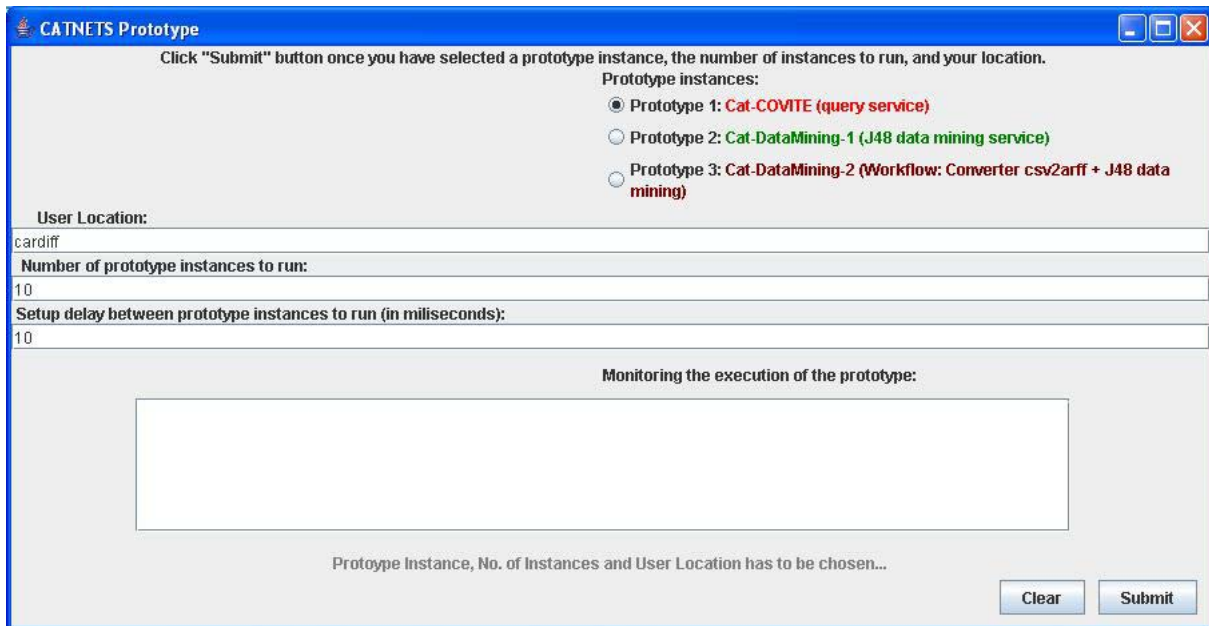
Figure 3.4 – Example of application prototype GUI with parameters

3.1.2 CATNETS Service Provider Advertisement Service

The following tool is part of the CATNETS prototype and allows service providers to advertise their services on the GMM market. This is the service needed by the application user.

Figure 3.5 presents the GUI interface of this tool. The following are the functionalities of the tool:

- FIND button helps to locate an agreement (search using Name (Service Name) field).
- ADD button helps to create a new agreement template -- storing the attributes in the 'AgreementTemplates' table and advertise the template in the 'agreementtemplate' table of cap_agreementtemplates_db database from the CAP repository. - UPDATE button helps to update the attributes of the agreement template (first use FIND button to locate the agreement you want to update), as well as the advertised agreement template.
- DELETE button helps to delete an agreement template record (first use FIND to locate the agreement you want to delete), in both tables: AgreementTemplates - that stores the template attributes, and in the agreementtemplate -- that stores the advertised agreement template
- CLEAR button to empty the fields.

The following parameters are needed:

- Name (Service Name) – the name of service advertised by the service provider, so that application users can search for. The following Service Names are already available within the CATNETS prototype (as services available on GMM): QueryService, J48DataMiningService, ConverterCsvToArff-J48DataMiningService, ConverterCsvToArff_Service, ContourService
- AgreementInitiator – the name of the service provider. Example: cardiff, barcelona, etc.

- ExpirationTime – defines the time until the service advertised in valid to. The "ExpirationTime" attribute has to be entered ONLY in this format: 'YYYY-MM-DD'
- BasicService Type – the name of the basic service that is made up the service available.
- PayForService – the maxim price (budget) the application user is willing to pay for the service from the market. The service provider can also fill in this parameter, meaning that the service is available for at least the price specified by this parameter, as minimum.
- ApplicationInstance and AgreementTemplateName are two parameters needed by the CATNETS prototype. In order to run the prototypes, the following names have to be used in the "ApplicationInstance": cat-J48DataMining_WS, cat-covite, cat-converterCsv2Arff_WS-J48DataMining_WS, converterCsv2Arff_WS, cat-ContourService.

Figure 3.6 – CATNETS Service Advertisement prototype

3.2 Middleware Layer Components

The dedicated resources model operates in the following manner in the context of the experimental setup for the Data-Mining prototype.

We setup controlled experiments by deploying several instances of the GMM in a Linux server farm. Each machine has a 2 CPU Intel Xeon at 2.80 GHz and 2 GB of memory. The nodes in the farm are connected by an internal Ethernet network at 100Mps. The topology is a

mesh; all nodes are interconnected to each other. CFPs are transmitted via groupcast to all the nodes in the destination groups (in our scenario CFPS are groupcasted from CSs to BSs).

We deploy the GMM in a number of nodes. Some nodes host a BS each and the Data Mining Web Service, while other nodes host the CSs, access points and clients. The Web Services are exposed in Tomcat servers. Access for execution of these Web Services on the resource node is what is traded between BSs and CSs. The experiments consist in launching 2 clients concurrently, which use each one of the CS as broker. Each client makes a number requests to the CS in intervals of a few seconds. Whenever a CS wins a bid with a BS, it invokes the Data Mining Service in the selected node, and the resource in the corresponding node gets locked for the duration of the service execution. We measure the selling prices of the BSs and observe the proportion of successful CFPs issued by the CSs.

4 Prototype performance measuring components for experiments

4.1 Application Scenarios & Use Cases

Application scenarios and uses cases have been presented in WP3-year 2 deliverable. This section presents the use case used for CATNETS tests made using the J48 Data Mining example. The scenarios are fully described in WP4-year3 deliverable.

The use case implemented within the CATNETS prototype has the following flow: an application 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 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 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 application prototype instances, based on the above scenarios, have been implemented:

- 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.

4.2 Performance measuring components for experiments

This section deals with the metrics collected at the agent layer of the middleware. First the process of executing a demand on the CATNETS market will shortly be sketched to set the context of the metric collection, for a more detailed discussion of this process see section 2.3.3 in this WP3 and the Deliverables for WP1. Afterwards the metrics collected will be presented along with the actual situation within the process when they are gathered.

Some metrics are semantically meaningful for all agents involved some only make sense for a subset of the involved agent types (ApplicationProxyAgents, CSA, BSA, ResourceAgents). All metrics collected will now shortly be described along with the individual points in the overall process when they are collected and the agent types collecting them.

Metrics collected in the Catallactic agents:

- Negotiation Request Metric

- Negotiation Start Metric

- Negotiation End Metric

- Strategy Metric

Each of these metrics represents a bundle of raw data that is semantically grouped. A more detailed description follows below.

Negotiation_Request metrics consist of the following data:

- Timestamp $\rightarrow$ timestamp stating when this particular metric was collected/written

- Writing Agent $\rightarrow$ Agent that was collecting/writing this metric

- NegotiationID $\rightarrow$ Identifier of the negotiation this metric is related to

This metric is always written when a new request for negotiation is initiated. This differs from ApplicationProxyAgents to all other agent types. The ApplicationProxyAgent writes this metric when sending the demand message to the CSAs. The CSAs and BSAs write it when sending a CfP to either the Service Market (CSA) or the Resource Market (BSA).

Negotiation Start metrics consist of the following data:

- Timestamp $\rightarrow$ timestamp stating when this particular metric was collected/written

- Writing Agent $\rightarrow$ Agent that was collecting/writing this metric

- "start" $\rightarrow$ flag stating that this metric is a negotiation start metric

- NegotiationID $\rightarrow$ Identifier of the negotiation this metric is related to

This metric is written after the discovery timeout is reached and a one-to-one negotiation is started (if at least one potential negotiation partner answered the CfP). Since the groupcast sent from the ApplicationProxyAgents to the CSAs does not result in a one-to-one negotiation between the ApplicationProxyAgent and the CSA – the chosen CSA just starts a CfP on the Service Market – this only CSA and BSA write the negotiation start metric.

Negotiation End metrics consist of the following data:

- Timestamp $\rightarrow$ timestamp stating when this particular metric was collected/written

- Agent 1   → Agent sending the accept/reject message

- Agent 2   → Agent receiving the accept/reject message; agent writing this metric

- "end"   → flag stating that this metric is a negotiation end metric

- NegotiationID   → Identifier of the negotiation this metric is related to

- "failure" or "success"   → flag stating the result of the negotiation

The negotiation end metric is written whenever an accept or reject message (stating the end of a one-to-one negotiation) is received. This metric is written by the receiving agent. Since in a one-to-one negotiation each of the both involved agents can end the negotiation (i.e. sending an accept or reject message), all agents of the CATNETS market (CSA, BSA and ResourceAgents) can write this metric.

The Strategy metrics consist of the following data:

- Agent   → writing agent

- "seller" or "buyer" negotiation   → flag indicating the role of the particular agent in this negotiation

- Acquisitiveness   → concession probability

- Price Step   → price step probability

- Price Next   → price jump between two negotiations

- Satisfaction   → probability to continue negotiation

- weightMemory   → weight of old negotiation results

- average Profit   → agent's fitness

- generation   → crossover counter

- currentMessageID   → message id

- currentMessagePrice   → price at which the negotiation was ended

- currentAverage   → estimated market price

- currentLowerLimit   → lower price range limit

- currentUpperLimit   → upper price range limit

The strategy metric is always written when a negotiation was ended successfully (accept message was sent). However this metric is written by both agents involved in this negotiation in order to log both strategy values. For clarity reasons only the important data used for calculating the overall metrics for the CATNETS market are indicated by additional

information here. For a more detailed overview of all strategy values used for implementing the agents' intelligence see WP1.

Based on this information the agent-related metrics can be calculated (see next section for a summary of all metrics used within CATNETS and WP4 for further information about the computation of higher level metrics).

4.3 Performance Results

Table 4.1 provides a global view of the metrics measured with the CATNETS prototype and the alignment with the metrics measured within the simulator.

We refer to WP4 year 3, section 3.8 for the experimental results and evaluation for the all three decentralized economic algorithms implemented in the prototype and its evaluation.

Metric	Description	Prototype Layer	Simulator Layer
start timestamp.	Experiment start time	APPLICATION (NOT NEEDED)	Simulator
end timestamp	Experiment stop time	APPLICATION (NOT NEEDED)	Simulator
Timestamp	Time elapsed from the beginning of simulation and experiment (Simulation time reference/Realtime for prototype).	MIDDLEWARE	AGENT
Number of Demand Requests	This metric counts the number of launched discovery processes until this contract is achieved.	APPLICATION	AGENT
Number of Negotiation Requests	This metric counts the number of launched negotiation processes until this contract is achieved	AGENT	AGENT
Number of Demand Accepts	This metrics counts the number of accepted demands before a transaction occurs	APPLICATION	AGENT
Number of Negotiation Accepts	This metrics counts the number of accepted negotiations before a transaction occurs	AGENT	AGENT
Agent Satisfaction	Calculated from 3 variables: Final price of the negotiation Reservation price for the buyer maximum price he is willing to pay; for the seller minimum sell price	AGENT	AGENT
Discovery Time	This metric is used to measure the time to find a given set of possible negotiation partners.	AGENT	AGENT
Negotiation/Waiting time	The measurement of the negotiation time starts after service discovery has completed, and ends before service usage or service provisioning. For centralized approach, this also comprises the allocation time	AGENT	AGENT
Service Provisioning Time	The evaluation framework defines the service provisioning time as the service usage time of one transaction (This metric is only taken into account for the prototype, as provision time cannot be fixed.).	APPLICATION This metric represents the ServiceProvisiongTime, in one transaction (not in one experiment). Stored in metrics_prototype repository. Normally it should happen thatService Provisioning = Service Allocation Time + Job Execution Time	Messaging Infrastructure
Service Allocation Time	Negotiating + discovery time	MIDDLEWARE negotiation Time.txt + discovery(constant time)	Messaging Infrastructure
Job Execution Time	The job execution time is defined as a sum of discovery time,	APPLICATION	MATLAB

	negotiation time (waiting time in centralized approach), network transfer time and execution time (which is - however - fixed for simulation).	Job execution time = executionTime.txt	
sender agent	Message transmitter	AGENT	MESSAGING INFRASTRUCTURE
receiver agent	Message receiver	AGENT	MESSAGING INFRASTRUCTURE
hops	Distance between the partners in hops	MIDDLEWARE Constant 1: Mesh topology	MESSAGING INFRASTRUCTURE
latency time	Distance between the partners in latency time.	MIDDLEWARE Constant for the cluster measured once	MESSAGING INFRASTRUCTURE
Agent id	The unique id of the agent.	AGENT	AGENT
Service Usage	The service usage is evaluated by the ratio between the service provisioning time and the total simulation time.	APPLICATION Calculated in a matlab script per each experiment service provision Time	AGENT
Message Size	Sum of Message Size for each transaction	MIDDLEWARE Constant size for the messages message size in the middleware Calculated once.	MESSAGING INFRASTRUCTURE
Number of Messages	This value counts the number of messages. (only the messages related to a negotiation will be count)	AGENT Count the number of messages in the experiments	MESSAGING INFRASTRUCTURE

Table 4.1 – Metrics alignment at prototype and middleware

5 Relation to other WPs

WP3 deals about the prototype, typically the practical implementation of all the concepts and tools from the rest of WPs. As such, WP3 has strong ties with the rest of WPs. From WP1 it takes the decentralized market design. The catallactic algorithms implemented in the GMM in WP3 follow the specifications from WP1. The scenarios for testing WP3 components have been aligned with those of the simulator from WP2 (cf. section 4.5 in WP3). Finally, all the results extracted from prototype experiments are evaluated using the analysis tools developed in WP4.

5.1 WP1

The objective of WP1 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 catallactic case is implemented as a resource allocation mechanism in the prototype. The results of the implementation of both centralized and decentralized mechanisms in simulator and the decentralized mechanism in

the prototype will demonstrate the applicability and feasibility of the strategies used in the markets.

5.2 WP2

The prototype demonstrate the feasibility of and ALN application based on Catallactic service/resource allocation - the decentralized case, while the simulator implements both centralized and decentralized cases, which permits to run large scale scenarios for extensive comparison of system performance. There are scenarios defined in order to run the experiments of which the results are compared and presented in WP4.

5.3 WP4

The work of WP3 provides as a result parts of the performance measuring infrastructure for the CATNETS prototype used in the measurements made within WP4. The view in WP3 is on implementation related aspects of the prototype for performance measurements, while in WP4 the focus is on the evaluation of the developed prototype in a wider sense, including its feasibility, aspects of generalization, standards used, performance in experiments, and identified limitations, leading as a result to a statement on implementing Catallactic-based resource allocation in application layer networks.

6 Economical model for eNVS

eNVS is a proposed experimental National Visualization Service (eNVS) for the UK e-Science programme [VizNET-LOUGH-0001-060811] .

6.1 Hypothesis

Our working hypothesis is that at present CATNETS has the expressive power to model and inform the establishment of a proposed eNVS. We can examine different case studies to explore this hypothesis under CATNETS.

Main question
How the CATNETS mechanism can be used for creating a price model, which can be applied for eNVS?

6.2 Scenarios

We consider two proposed eNVS scenarios, presented below, and map them to the CATNETS scenarios:

- batch mode

This scenario is about "LOCALITY", and requires a linear visualization pipeline where all the resources are present at the same location and it can be mapped to the first scenario from CATNETS -- Single Service. The batch mode scenario is presented in Figure 6.1. Symbols "A", "B" and "C" in Figures 6.1 and 6.2 represent services involved in a workflow.
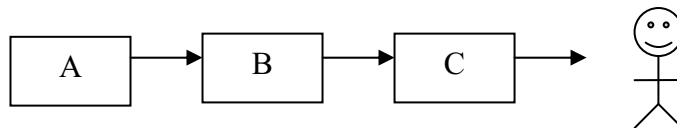
Figure 6.1 - Batch mode of process execution

- interactive mode

This scenario, presented in Figure 6.2, is about "PERSISTENCY", and requires distributed resources, which could be based at different locations. It also requires co-allocation of resources; therefore the service will only start if all resources have been reserved in advance. The resources will be released after all the jobs have been completed. After the service is completed, the user may need to re-use some or all of the resources, which leads to several possible feedback actions to occur in this process. This scenario can be mapped to the second scenario from CATNETS – Multiple Service.
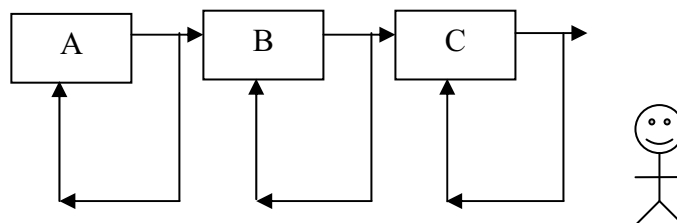
Figure 6.2 - Interactive mode of process execution

One main gap in the existing CATNETS approach has been identified after mapping our visualization scenarios to the application scenarios developed in CATNETS; namely, that the CATNETS service is not interactive during the job execution, therefore it cannot provide currently the required interactivity for some visualization services, e.g. the need to restart the service or to change parameters dynamically. Such capability is essential to support interactive visualization applications, as exemplified through the general area of "computational steering". In such an approach, visualization is separated geographically from a machine running the analysis or simulation. Any change in simulation parameters causes data to be transferred over the network to the user to be visualized. Such interaction between a visualization and simulation engines cannot be undertaken effectively using a batch mode execution.

6.3 Algorithm for decision making regarding the hypothesis
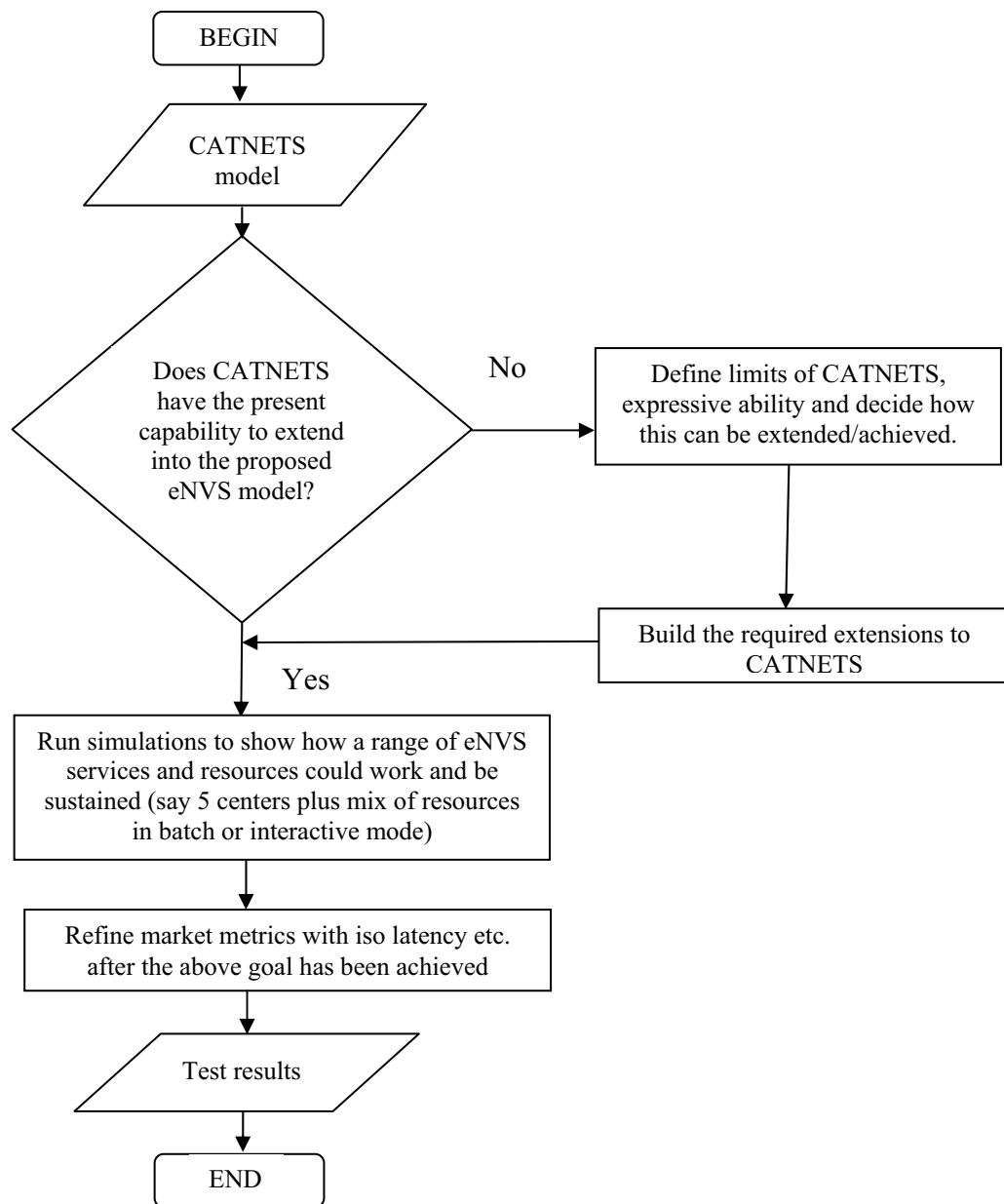
Figure 6.3 - Algorithm for decision making

6.4 Hypothesis testing

For the purpose of testing the hypothesis, an example of visualization Web Services, such as Marching Cubes services and the Contour Filter service, developed by G. Shu and N. Avis [Shu07] need to be run on the CATNETS framework and the results visualized. It is very important to determine the difference between these results and the results from the Data Mining example, which is a case study provided in CATNETS.

Shu and Avis [Shu07] present a novel workflow-based approach to the design and implementation of distributed visualization systems in a Grid computing environment. The approach evolves the traditional dataflow visualization system into an environment in which visualization Web Services can be linked with local visualization tools to form a pipeline using workflow technology. A specific aim of our design is to provide a distributed visualization toolkit to fulfil a user's request for any particular visualization techniques/algorithms, allowing users to select the most appropriate resources, and compose the visualization pipeline.

Each visualization algorithm available in the Visualization Toolkit (VTK)[1] was developed into a Web Service. For example, the Web Service "MarchingCubes" is for the algorithm "Marching Cubes". As Web Services, visualization services are wrapped using WSDL (Web Service Description Language) and SOAP (Simple Object Access Protocol) interfaces and messaging. These visualization services serve to filter data, provided by source, into geometry. For example, the service "ContourFilter" processes data by means of a contour algorithm. It takes as input any dataset and generates on output isosurfaces and/or isolines. These visualization services are developed using ij-VTK, which is a Java wrapper for VTK. Taking the service "ContourFilter", mentioned above, for instance, it uses vtkPolyDataWriter to write output of vtkContourFilter, which is polygon data, into a binary data file in VTK format, and sends the file as a SOAP message to the client-side application. On the other hand, the client-side application will use vtkPolyDataReader to read the binary polygonal data file in VTK format.

Below is a case study presented as a proof-of-concept demonstrator. The case study shows how the various visualization tools in the Triana workflow system can be used in conjunction with the visualization Web Services to form a visualization workflow. Figure 6.4 illustrates the composition tools provided for the visualizing workspace. On the left hand side, the user is provided with a collection of pre-defined folders containing tools grouped according to function. The tools in the DataInput folder, for example, perform tasks inputting local dataset and remote dataset. On the right hand side, the user is provided with workspace for composing workflows by dragging and dropping the units/tools from the toolboxes. Each tool or unit represents a service with a pre-defined input/output interface. The connection between tools is made by dragging a cable from the output node (right-hand side) of the sending task to the input node (left-hand side) of the receiving task. Once a workflow has been created it can be executed. A Web Service is imported to the workspace by providing its WSDL interface. If the interface is provided, Triana creates a tool for each operation provided by the service. These tools are used to invoke the service operations and are similar to the pre-defined tools but have a different colour in the workspace. The toolbox, which provides additional support for creating a workflow and using the developed visualization Web Services, is in the Visualization folder. A variety of visualization Web Services, such as MarchingCubes, ImageFilter, which are deployed on a Windows platform and registered, at

---

[1] *http://www.vtk.org/*

this moment, in our private UDDI and have been imported into Triana, are shown in the WebServices folder.

The data used in the case study is obtained from [Schroeder03]. It deals with Computed Tomography (CT) data. This study contains 93 such slices, each of which is taken perpendicular to the spine approximately through the middle of the ears, spaced 1.5 mm apart. Each slice has 2562, 0.8 mm pixels with 12 bits of gray level. We build a workflow with two branches, one branch for using algorithm "ContourFilter" to filter data, and the other "MarchingCubes". Two branches work in parallel in order that we can judge the different rendered results from different algorithms as soon as possible. The workflow first inputs the data set from remote machine by means of tool "RemoteDataSet", then reads the data set and converts it into the VTK format. In the end, the workflow filters the data set using Web Service "ContourFilter" and "MarchingCubes" respectively and renders them. The test shows that "MarchingCubes" is faster than "ContourFilter", but the resulting images are almost the same.
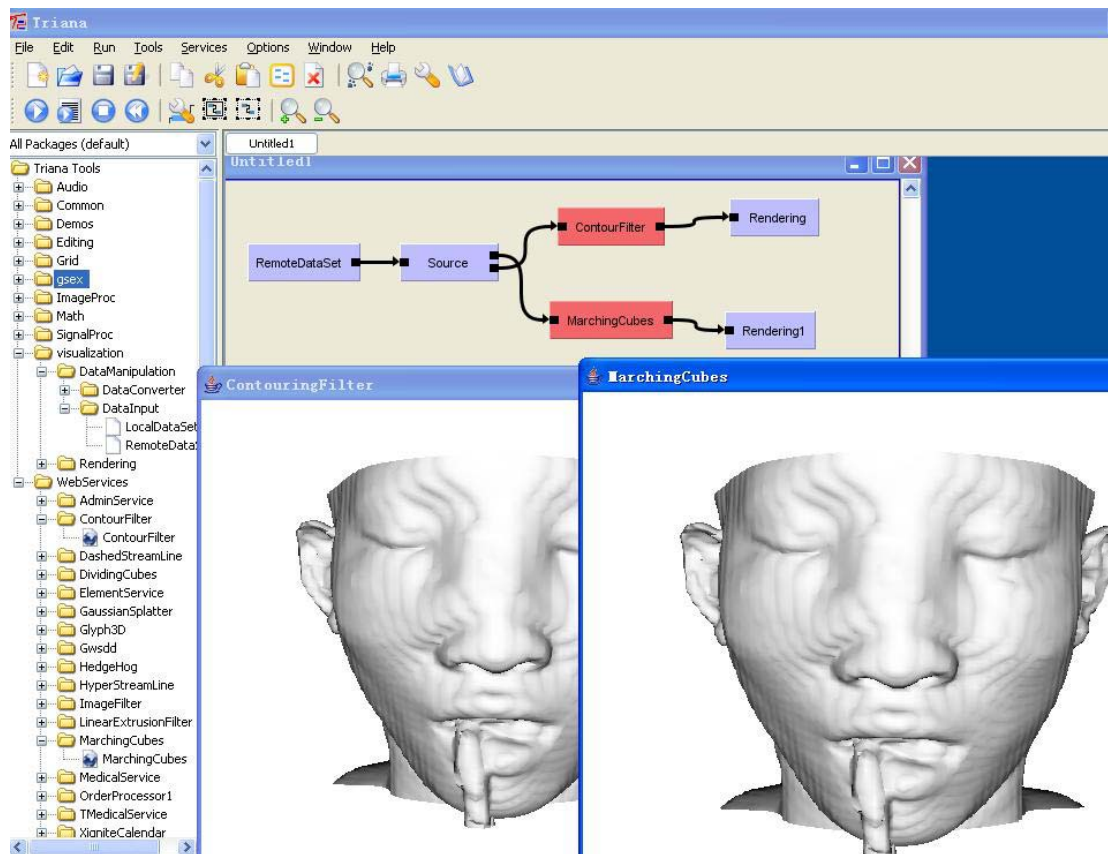
Figure 6.4 - A case study for workflow-based distributed visualization

6.5 General observations about CATNETS in regard to the hypothesis and the main question presented above

CATNETS provides some support for initiating an interactive service, because the user can interact with the services while they run. Initially, the process of negotiation for the resource is an interactive process, where the customer and the resource provider negotiate in interactive mode about the time, the price and the location of the required resources. Once the resource has been allocated to a certain user, the user can start his job in a batch mode, but he can still

interact with the service while the job is running, getting up-to-date information about the state of the job, the estimated time left etc.

The main question is: How much to charge for the service? To be able to answer this question, we need to determine the process parameters, business rules and to calculate the cost of the service based on this data. We will also need to model the business processes in the middle layer of CATNETS, as well as the ones in the Application layer. There is a need for a business model of the Application layer, so we can calculate the cost of the service.

The main parameters of the services in CATNETS are:
1. Number of requests (number of instances you run the application)
2. The delay between each request
3. Budget – as a fixed parameter (should be calculated as a cost)
4. Resource parameter
5. Time (needs to be calculated – how long the user is taking the resource for)
6. The initial cost of the resource
7. Location parameter showing where the resource is located (using MySQL repository on one machine, CAP (Catallactic Access Point) on another machine)
8. Price offered by the application layer – it is fixed currently

There is a request on demand – if a user requires a resource, he sends a request and receives an offer back with the price and the location of the resource. If he agrees to use the resource for this service, he sends his URI back and can start his job. If the user does not accept, he simply does not send anything back.

There is also an archive of metrics, agreement template, agreement offer, module MGS (Master Grid Service) running on the service. The metrics are: service provisioning time (the total time of the process), job execution time and ID of the experiment. Another useful parameter is the Number of basic service nodes.

In Figure 6.5 a possible scenario is presented, when a user requests a service, which can be performed at different locations on distributed resources. The figure shows example services A, B and C, which could represent the Data Input, Contour Filter and Rendering Services within Figure 6.4, respectively. These services have been arranged using quality of service (QoS) criteria, such as a notion of iso-latency, which corresponds to the time to receive a response once a request has been sent to any one of these services. These services have been grouped according to some defined QoS thresholds and as such the price for each service can theoretically depend on QoS metrics.

Figure 6.6 represents the intersection between the batch and the interactive services, united by the CATNETS approach. The J48 data mining example is shown as a batch processing application. This reflects the UK's National Grid Service (NGS) operation [NGS]. The proposed eNVS is an adjunct to the NGS that must support both batch and interactive services. Therefore, we need to test the market in both ways – for batch and interactive mode – and to estimate which one is more economically efficient from the provider's point of view.
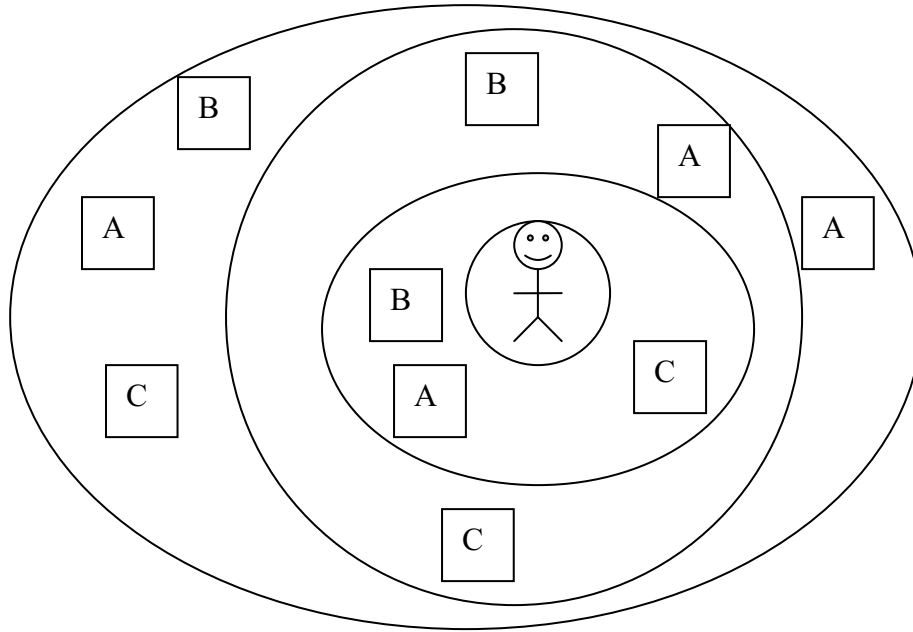
Figure 6.5. Groupings of services based on QoS criteria
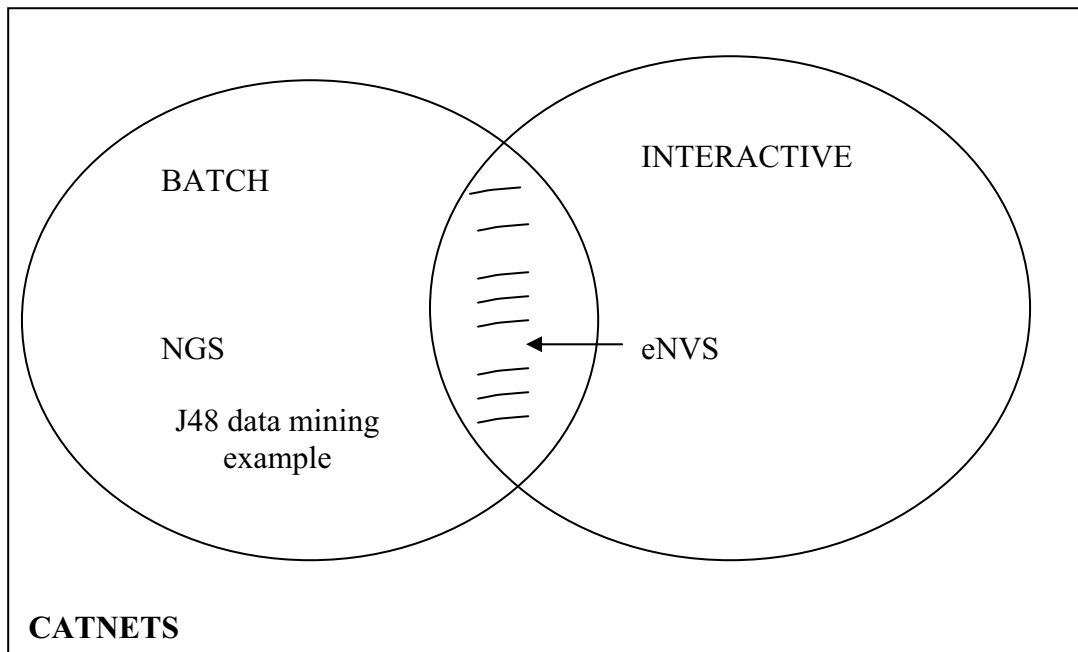
Figure 6.6 - The intersection between Batch and Interactive services

The price estimation in CATNETS depends on the resource price. However, there is a lower bound to this price, this threshold must never be exceeded. This is valid for the batch mode market. The interactive mode has not been incorporated in the existing CATNETS model.

It is possible to view the second scenario (Figure 6.2) as two concurrent CATNETS markets – batch and interactive. CATNETS has the concept of the resource bundle already, which can be refined and used in the case of interactive mode.

If we consider the second scenario as a portfolio of services, we need to consider Price Risk Management. Another possible solution for the interactive mode could be to use Decentralized Catallactic approach, where the user requests two or three times the time slot, which he would have requested for the batch mode and he does not care about location, time, resource etc.

We have to use the CATNETS scheduler to price these two models – batch and interactive. There is no need to change the scheduler for this purpose.

We can model the services in two different ways and explore the market twice as a batch or interactive service. For this purpose, the Time should be added to the current model as a parameter.

It is also important to be able to determine the number of service instances to support in these two markets (batch and interactive). Market intervention approaches (for instance, promoting interactive services ahead of batch mode services) will be explored to better understand this problem.

6.6 Conclusions and further directions

There are three main things to be done:
1. We need to define a time parameter in the resource.
2. What request you make through CAP (Catallactic Access Point) to SLA (Service Level Agreement) request? – need to change the SLA for the interactive mode.
3. How to price both of these models (batch and interactive) based on the scheduler?

We will conduct further simulation results with a view to informing the incorporation of the CATNETS model into a future eNVS. The outcomes of these studies will be reported by means of presentations, reports and talks. Another expected outcome is a publication in a journal or conference.

7 Conclusions of WP3

7.1 Prototypes using Catallactic-based markets

The implementation of two prototypes with a Catallactic based mechanism, as well as a generalisation of application with the GMM, detailed in WP3 year 2 deliverable and enhanced with more details within this deliverable, have raised interesting conclusions outlined below:

- COVITE prototype [COVITE, 2007], [Joita, 2004] supports the creation of Virtual Organisations (VOs) via a centralised system. All resources and services are allocated by a central authority at the creation of the VO. A decentralized solution, as Cat-COVITE prototype presented above, in which services and resources are to be allocated in peak/busy times to application users / processes is an straightforward way to deal in such demanding situations, while there is no other possibility of adding more resources or the availability of services are limited.

- A second solution, as Cat-Data Mining prototype, has been proposed and developed. The main problem 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 Catallactic markets expect to provide members of VOs a fair price for services needed by the users' application.

- The Catallactic markets also expect to provide a fair welfare distribution among the participants, such as users' application, and service and resource providers.

- 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.

- Relevant prototype metrics are proposed and a composite index - social utility factor, is being calculated as a single index, so that comparison and evaluation of different scenarios, as well as between prototypes and simulators can be realised. The process of extracting and analysing the results are shown in Deliverable WP4.

References

[Ardaiz, 2005] O. Ardaiz, P. Chacin, I. Chao, F. Freitag, L. Navarro, An Architecture for Incorporating Decentralized Economic Models in Application Layer Networks, Internacional Workshop in Smart Grid Technologies, Utrecht, Holanda, July 25 - 29, 2005.

[Chao, 2004] Isaac Chao,Ramon Sangüesa and Oscar Ardaiz, Design, Implementation and Evaluation of a Resource Management Multiagent.System for a Multimedia Processing Grid, Workshop on Grid Computing and Its Application to Data Analysis (GADA) On the Move to Meaningful Internet Systems 2004

[COVITE, 2007] COVITE project. http://www.wesc.ac.uk/projectsite/covite/index.html

[Joita, 2004] 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

[Despotovic, 2004] Z. Despotovic, J.-C. Usunier, K. Aberer: Towards Peer-To-Peer Double Auctioning, Proceedings of the 37th International Hawaiian Conference on System Sciences (HICSS), Waikoloa, Hawaii, USA, 2004

[FIPA, 2007] FIPA webpage: http://www.fipa.org/

[Foster, 2002] Foster, I.; Kesselman, C.; Nick, J.M.; Tuecke, S., Grid services for distributed system integration, Computer, Vol.35, Iss.6, Jun 2002, Pages:37-46

[NGS] Neil Geddes, Andrew Richards, Malcolm Atkinson, Steve Chidlow, W. T. Hewitt, Stephen Pickles, Paul Jeffreys, "Workshop on Sustainability and Future Business Models for the UK National Grid Service", National e-Science Centre, Edinburgh, May 2007, http://www.nesc.ac.uk/technical_papers/UKeS-2007-05.pdf

[Schnizler, 2006] Schnizler, B., Neumann, D., Veit, D., Weinhardt, C., "A Multiattribute Combinatorial Exchange for Trading Grid Resources", Proceedings of the 12th Research Symposium on Emerging Electronic Markets, Amsterdam, 2005

[Paurobally, 2004] Shamimabi Paurobally, Jim Cunningham, and Nicolas R. Jennings. Verifying the contract net protocol: A case study in interaction protocol and agent communication language semantics. In Proceeding 2nd International Workshop on Logic and Communication in Multi-Agent Systems, 2004

[Ogston, 2002] E. Ogston and S. Vassiliadis, A peer-to-peer agent auctionProceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)

[Preist, 1998] 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

[Schroeder, 2003] William Schroeder, Ken Martin, Bill Lorensen, The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics (3rd Edition), Publisher: Prentice Hall; (2003).

[Shu, 2007] Gao Shu, Nick J. Avis, Workflow-Based Distributed Visualization, 2nd International Workshop on Workflow Management and Application in Grid Environments (WaGe07), August 16-18, 2007, Urumchi, Xinjiang, China

[Streitberger, 2007] Werner Streitberger, Torsten Eymann, Daniel Veit, Michele Catalano, Gianfranco Giulioni, Liviu Joita, Omer F. Rana - "Evaluation of Economic Resource Allocation in Application Layer Networks - A Metric Framework", 8. Internationale Tagung Wirtschaftsinformatik -- eOrganisation: Service-, Prozess-, Market-Engineering, 28 February - 2 March 2007, Karlsruhe, Germany.

[VizNET-LOUGH-0001-060811] R.S. Kalawsky, "A Visualization Service for the National Grid Service: A Workshop to derive User Needs", Loughborough University, VizNet, 18 May 2006, http://www.nesc.ac.uk/technical_papers/UKeS-2006-05.pdf

[WP3-Year2, 2006] Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freitag, Liviu Joita, Manuel Medina, Leandro Navarro, Omer F. Rana, Miguel Valero, "WP3, Year 2 Catnets deliverable", August 2006

[WP4-Year1, 2005] Michele Catalano, Gianfranco Giulioni, Werner Streitberger, Michael Reinicke, Torsten Eymann, WP4, Year1 Deliverable, August 2005

[WS-Agreement, 2007] Web Services Agreement Specification (WS-Agreement), https://forge.gridforum.org/sf/projects/graap-wg

Annex A – CATNETS Repositories Settings

A. 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

Below are the steps needed to creating an instance of a supplier database:

- Creating 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');
```

- The way to check the data entries is by running the query below:
```
mysql> SELECT * FROM SupplierDB.Product;
```

The following script performs the operations describe above:
```
CREATE DATABASE SupplierDB;
use SupplierDB;
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)
);
INSERT INTO Product VALUES (NULL,'manufacturer1','chair1','10'),
(NULL,'manufacturer2','chair2','15'), (NULL,'manufacturer3','chair3','20'),
(NULL,'manufacturer4','chair4','25'), (NULL,'manufacturer5','chair5','30');
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.

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

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;
```

MySQL SCRIPTS: The following scripts perform only the operations needed above, in case of location "cardiff":
CREATE DATABASE loader_db;
use loader_db;
CREATE TABLE loader_db.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)
);
INSERT INTO Loader_DB.UserLoaderJobs VALUES (NULL,'cardiff','cat-covite','SELECT * FROM SupplierDB.Product ORDER BY Price DESC;'),(NULL,'cardiff','cat-J48DataMining_WS','http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.arff'),(NULL,'cardiff','cat-converterCsv2Arff_WS-J48DataMining_WS','http://users.cs.cf.ac.uk/L.Joita/dataFiles/weather.csv');

SELECT * Loader_DB.UserLoaderJobs;

- As a "root" (administrator of the database) account, create user "catnets" with password "catnets" that has all privileges on the database Loader_DB.

```
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;
```

MySQL SCRIPT: The following script performs only the operations needed above: cap_urls_db-script.sql (***BEFORE*** running this script, read the following file: cap_urls_db-script-sql-README.txt)


```
CREATE DATABASE CAP_URLs_DB;
use CAP_URLs_DB;
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)
```

```
);
INSERT INTO CAP_URLs_DB.CAP_URLs VALUES
(NULL,'cardiff','http://131.251.128.7:18088/axis/CatallacticAccessPoint/Cat
allacticAccessPoint.jws');
# ENTER your relevant address for the Catallactic Access Point web service
# INSERT INTO CAP_URLs_DB.CAP_URLs VALUES
(NULL,'barcelona','http://<ipaddress>:<port>/<path-to-CAP>');
# INSERT INTO CAP_URLs_DB.CAP_URLs VALUES
(NULL,'bayreuth','http://<ipaddress>:<port>/<path-to-CAP>');
# check the data entry
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.


```
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,
        `AgreementTemplatesReference` SMALLINT UNSIGNED NOT NULL DEFAULT 0,
        `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`)
        CONSTRAINT `FK_AgreementTemplatesReference` FOREIGN KEY
`FK_AgreementTemplatesReference` (`AgreementTemplatesReference`)
          REFERENCES `AgreementTemplates` (`ID`)
          ON DELETE RESTRICT
          ON UPDATE RESTRICT
      )
      )
     ENGINE = InnoDB
     COMMENT = 'Contains agreement templates of the complex services
available at this Catallactic Access Point location';
```

There is no need to insert data into this repository. For this, use CATNETS WS-Agreement Prototype to insert (advertise) the services at CAP repository – see section 3.1.2
/* 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;
COMMENT = 'Contains agreement offers received from the MGS (application
level)';
```

The following script performs only the operations needed above: cap-agreementtemplates-script.sql

```
CREATE DATABASE CAP_AgreementTemplates_DB;
use CAP_AgreementTemplates_DB;
CREATE TABLE `cap_agreementtemplates_db`.`agreementtemplate` (
  `ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `AgreementTemplatesReference` SMALLINT UNSIGNED NOT NULL DEFAULT 0,
  `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`)
  CONSTRAINT `FK_AgreementTemplatesReference` FOREIGN KEY
`FK_AgreementTemplatesReference` (`AgreementTemplatesReference`)
    REFERENCES `AgreementTemplates` (`ID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT
)
ENGINE = InnoDB
COMMENT = 'Contains agreement templates of the complex services available
at this Catallactic Access Point location';
```

```
SELECT * FROM cap_agreementtemplates_db.agreementtemplate;
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
COMMENT = 'Contains agreement offers received from the MGS (application
level)';
#SELECT * FROM cap_agreementtemplates_db.AgreementOfferReceived;
CREATE TABLE `cap_agreementtemplates_db`.`AgreementTemplates` (
  `ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL DEFAULT '' COMMENT 'Agreement Name',
  `AgreementInitiator` VARCHAR(45) NOT NULL DEFAULT '',
  `ExpirationTime` DATE NOT NULL,
  `BasicServiceType` VARCHAR(45) NOT NULL DEFAULT '',
  `NumberOfBasicServiceNodes` INTEGER UNSIGNED NOT NULL DEFAULT 1,
  `PayForService` INTEGER UNSIGNED,
  PRIMARY KEY(`ID`)
)
ENGINE = InnoDB
COMMENT = 'Store the agrement template elements for CATNETS prototype';
```

- Create a table named AgreementTemplates within the CAP_AgreementTemplates_DB. This table hosts the agreement template elements.

```
mysql> CREATE TABLE `cap_agreementtemplates_db`.`AgreementTemplates` (
  `ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL DEFAULT '' COMMENT 'Agreement Name',
  `AgreementInitiator` VARCHAR(45) NOT NULL DEFAULT '',
  `ExpirationTime` DATE NOT NULL,
  `BasicServiceType` VARCHAR(45) NOT NULL DEFAULT '',
  `NumberOfBasicServiceNodes` INTEGER UNSIGNED NOT NULL DEFAULT 1,
  `PayForService` INTEGER UNSIGNED,
  PRIMARY KEY(`ID`)
)
ENGINE = InnoDB
COMMENT = 'Store the agrement template elements for CATNETS prototype';
```

- As a "root" (administrator of the database) account, create user "catnets" with password "catnets" that has all privileges on the database CAP_AgreementTemplates_DB.
```
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;
```

MySQL SCRIPT: The following script performs all operations needed: mgs_agreement_db-script.sql

```
CREATE DATABASE MGS_Agreement_DB;
use MGS_Agreement_DB;
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.

```
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. Cat-DataMining prototype

1. Users' Loader Jobs Repository

Using the database already created at section A.2 -- `loader_db`, below are the 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;
```

MySQL SCRIPTS (already done at point A.2): The following scripts perform all operations of databases creation needed: loader_db-cardiff-script.sql, loader_db-barcelona-script.sql, loader_db-bayreuth-script.sql

Assign the user "catnets" and password "catnets" – THIS HAS TO BE DONE AFTER YOU RUN THE SCRIPTS (already done at point A.2)

2. Catallactic Access Point URLs Repository
Follow section A.3.

3. Catallactic Access Point Repository

Catallactic Access Point Repository hosts the agreement templates (in AgreementTemplate table), agreement offers (in AgreementOfferReceived) sent by the MGS, as well as hosts the agreement template elements (in AgreementTemplates).

There is no need to insert data into this repository. For this, use CATNETS WS-Agreement Prototype to insert (advertise) the services at CAP repository – see section 3.1.2

/* 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 of databases creation needed: cap-agreementtemplates-script.sql
Assign the user "catnets" and password "catnets" – THIS HAS TO BE DONE AFTER YOU RUN THE SCRIPTS (already done at point A.4)

4. Converter .csv format to .arff format

This is used for the Triana workflow.

4.1 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');
```

MySQL SCRIPT: loader_db-cardiff-script-1.sql

CAP Repository

There is no need to insert data into this repository. For this, use CATNETS WS-Agreement
Prototype to insert (advertise) the services at CAP repository – see section 3.1.2
```
/*
mysql> INSERT INTO cap_agreementtemplates_db.agreementtemplate VALUES
(NULL, 'converterCsv2Arff_WS','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>');
*/
```
MySQL SCRIPT: cap-agreementtemplates-script-1.sql

C. 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,
  `Experiment_ID` INTEGER UNSIGNED COMMENT 'Represent the ID of the
experiment',
  `ServiceProvisionTime` BIGINT UNSIGNED COMMENT 'Represents the time
required for MGS to complete a user request
(discovery+negotiation+execution). 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;

mysql> CREATE TABLE `metrics_prototype`.`metrics` (
  `ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Experiment_ID` SMALLINT UNSIGNED COMMENT 'Represent the ID of the
experiment',
  `ServiceProvisionTime` BIGINT UNSIGNED COMMENT 'Represents the time
required for MGS to complete a user request
(discovery+negotiation+execution). 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.',
  `Description` VARCHAR(200) DEFAULT '',
  `Approach` VARCHAR(200) DEFAULT '',
  `StartTimestamp` TIMESTAMP DEFAULT '0000-00-00 00:00:00',
  `EndTimestamp` TIMESTAMP DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY(`ID`)
)
ENGINE = InnoDB;
```

MySQL SCRIPT: The following script performs only the operations needed above:
metrics_prototype-script.sql

- As a "root" (administrator of the database) account, create user "catnets" with password
"catnets" that has all privileges on the database MGS_Agreements_DB.
```
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 B – GMM CATNETS FINAL RELEASE DESCRIPTION & USER MANUAL

What is on the tar file distribution[2]

The middleware tar consist of the following (all data store in the REPOSITORY, default is GMM_all)

No addtional exteral resources (programs, libraries ,etc) are required. All is cointained in the package and is automatically deployed by the deploymnet scripts.

*** Vxargs (included in the bundle) performs remote ssh on a set of pararell machines, allowing for a paralell deployemt on N machines from a single node. Vxargs requires Python in your system in order to work.

- middleware jars:
1 jar with the GMM for zip agents: middlewareBaseline.jar
2 jars with the GMM for catallactic agents: middlewareCS.jar and middlewareBS.jar
required libs stored in middleware/lib

- configuration
1 GMM congifuration file for the server nodes ZIP agents:
BSHostingConfig.properties
1 GMM congifuration file for the client nodes ZIP agents:
CSHostingConfig.properties
1 GMM congifuration file for the server nodes CNet agents:
CNetBSHostingConfig.properties
1 GMM congifuration file for the client nodes CNet agents
CNetCSHostingConfig.properties
1 GMM congifuration file for the server nodes catallactic agents:
CatllacticBSHostingConfig.properties
1 GMM congifuration file for the client nodes: catallactic agents:
CatallacticCSHostingConfig.properties
1 profile.xml for JXTA configuration
Files for catallactic agents detailed config:
complexService.properties
basicService.properties
resourceAgent.properties
strategy.cong
learning.conf

- required sofware and libs
apache tomcat as Web Server: apache-tomcat-5.5.12.zip
apache axis for Wed Services: axis-bin-1_3.zip
required libs to run Web Services: activation.jar and mail.jar

---

[2] All paths in the document are relative to the REPOSITORY PATH, by default GMM_all

vxargs script to deploy run on several machines in paralell and collect results

- Remote nodes lists:
nodeList.txt : list all nodes involved in the experiment
nodeListBS.txt : list all servers
nodeListCS.txt : list all clients

- scripts to execute from the base node (deployer)
1 script to deploy the middleware: deployMiddleware.sh
1 script to deploy small realease/config into the middleware: deployLittle.sh
1 script to run a catallactic experiment: runExperiment.sh
1 script to run a ZIP agents experiment: runZIPExperiment.sh
1 script to stop any running experiment: stopExperiment.sh
1 script to collect the metrics and logs for catallactic agents: collectMetrcis.sh
1 script to collect the metrics and logs for ZIP agents: collectZIPMetrcis.sh
1 script to clean all nodes in the cluster/LAN: cleanAllNodes.sh
1 script to be remotely executed in each node taking part in the experiment (this is done
automatically by the runExperiment.sh script): launchMidleware.sh

- Data Mining Services
classes and data for J48 Web Service deployment
1 class to act as J48 Web Service client

Vxargs is used to provide parallel execution of deployment and experiments

How to deploy the GMM

How to deploy/run on a given cluster or set of machines in LAN (using provided scripts)

All the commands are run do from the central node where you untared the package, thereafter
REPOSITORY_LOCATION, by default: GMM_all

Deploy from scratch

In scratch, run the following:
This deployment is required whenever a new node is added to the experiments, in order to
deploy java, tomcat and axis in the new node.

1 - edit deployMiddleware.sh and set REPOSITORYPATH and YOURPATH to the
convenient path on the cluster (default should work)
 REPOSITORYPATH -> The path where the GMM was untared, this is the central node for
deployment/execution
 YOURPATH -> Path on the remote nodes, this is the path where the GMM will be deployed
in the remote nodes selected in nodelist.txt

 2 - edit nodelistBS.txt , nodelistCS.txt and nodelist.txt with the list of nodes for the
experiment (see
    description for more info)

3 - edit CatallacticCSHostingConfig.properties and CatallacticBSHostingConfig.properties to config CSs and BSs (or keep on defaults)

    minimun: edit the config.workdirectory to a full path valid to your system; without this JXTA wont work

 4 - Deployment: this will deploy the middleware and required data in all nodes in nodeList.txt

    from the REPOSITORY location run:

    %>./deployMiddleware.sh (alternative %>source deployMiddleware.sh)

    If all exit codes of the vxargs operations output are set to 0, then you are should the GMM is correctly deployed

    if you get any exit code not 0, the errors. Go to ../deployment where all deployment output results is stored

This is for the catallactic agents. For the ZIP agents experiments do the same, but using CSHostingConfig.properties and BSHostingConfig.properties; for the CNetAgents do the same but using %> CNetCSHostingConfig.properties and CNetBSHostingConfig.properties

Deploy after config changes / GMM jar update

You don't need to deploy again the all the infrastructure whenever you make changes to agent's configurations or update the GMM jars. In this case run deploymentLittle.

5 - Little Deployment: this will deploy the middleware jar and configFiles in all nodes in nodeList.txt, nodeListCS.txt/nodeListBS.txt

    from the REPOSITORY location run:

    %>./deployLittle.sh (alternative %>source deployMiddleware.sh)

    If all exit codes of the vxargs operations output are set to 0, then you are should the GMM is correctly deployed

    if you get any exit code not 0, the errors. Go to ../deployLittle where all deployment output results is stored

How to run the experiments[3]

Experiment configuration

Whenever you change agents configuration files for a new experiment, you need to run deployLittle.sh in order to make the remote nodes aware of the new config for the experiment.

ZIP agents configuration is controlled in :
BSHostingConfig.properties
CSHostingConfig.properties

CNet agents configuration is controlled in :
CNetBSHostingConfig.properties

---

[3] No simultaneous experiments can be performed in a set of nodes: Just one experiment at a time.

CNetCSHostingConfig.properties

Catallactic Agents configuration is controlled in:
CatllacticBSHostingConfig.properties
CatllacticBSHostingConfig.properties
complexService.properties
basicService.properties
resourceAgent.properties
strategy.cong
learning.conf

Running Experiments

All the commands are run do from the central node where you untared the package, thereafter REPOSITORY_LOCATION, by default: GMM_all

6 - runExperiment : this will lauch a GMM instances in Servers and Client with several BSs(listed in nodeListBS.txt and CSs(listed in nodeListCS.txt) from the REPOSITORY location (default is GMM_all) run:
   %>./runExperiment.sh
    You may check Result live of vxargs executions .out and .err for all the nodes in ../outputVxargs
    resutsBS and resultsCS directories contains the execution output (experiment may last from 2 min to 10 or more, depending how many CS are lauched).

This is for the catallactic agents. For the ZIP agents experiments do the same, but using %> ./runZIPExperiment.sh, and for the CNetAgents do the same but using %> ./runCNetExperiment.sh

You may collect your metrics in any moment, this wont stop the experiment. You can collect them more than once, each time you do so a tar file with your metrics, logs up to that moment as well as the experiment config will be stored in ../results/resultsHistory

 7 - Stop experiment and cleanup.
    from the REPOSITORY location (default is GMM_all) run:
   %.>/stopExperiment.sh

 - Don't forget to stop the experiment after your metrics have been collected since this leaves alive processes on your system.

How to collect metrics

Raw Metrics available

The metrics from a prototype execution are stored in raw text files. For the ZIP agents and for the catalectic agents we get respectively the following metrics

Metrics for ZIP agents

METRIC NAME	METRIC VALUE
Price.txt	current price of an agent
Match.txt	shows the price at which an offer is accepted (offer+bid/2).
Active.txt	shows if the CS is in the market or not. After a successful match an agent leaves the market and returns with the probability of for example 1/3

By counting the events in Match.txt you and dividing by the total number of requests issued till the moment of metric collection, you can calculate the allocationRate.

Metrics for CNet agents

METRIC NAME	METRIC VALUE
Price.txt	current price of an agent
sellSucces.txt	Shows the number of succesfull trades

By counting the events in sellSuccess and dividing by the total number of requests issued till the moment of metric collection, you can calculate the allocationRate.

Metric for catallactic Agents

negotiation_start, negotiation_request, negotiation_end
strategy_metric

METRIC NAME	METRIC VALUE
negotiation_request.txt	A negotiation is requested between 2 agents (== CFP received by the BS?)
negotiation_start.txt	A negotiation starts between 2 agents
negotiation_end.txt	A negotiation ends between 2 agents
strategy_metric.txt	Contains the catallactic strategy values in this moment + the current price

The strategy metric contains the following fields, separated by commas:
Strategy_Metric detailed parameter of the agent strategy:

Agent
 Acquisitiveness
 Price Step
 Price Next
 Satisfaction
 weightMemory

averageProfit
generation
currentMessageID
currentMessagePrice
currentAverage
currentLowerLimit
currentUpperLimit

CurrentMessagePrice contains the current price in the negotiation.

In order to calculate the allocation rate, you need to count all the negotiation_end events containing a CS in the second or third positions. You need to take into account both the negotiation_end metrics fired by CSs or BSs. This happens since both CSs or BSs can close the negotiation.

Collecting the metrics

All the commands are run do from the central node where you untared the package, thereafter REPOSITORY_LOCATION, by default: GMM_all

The metrics are collected in ../results/metrics. You may collect your metrics in any moment, this wont stop the experiment. You can collect them more than once, each time you do so a tar file with your metrics, logs up to that moment as well as the experiment config will be stored in ../resultsHistory. Each of this metric collection is marked with the system timestamp unambiguously labelling the experiment results

8 - Collect logs and metrics: this will collect all metrics and logs for the experiment logs, and store a tar with all your experiment results in the RESULTSH_HISTORY_PATH. Provide your user identification to properly label your experiment.
     If you changed the default YOURPAHT, then update also YOURPATH in the collectMetrics.sh script
     metrics and logs for an experiment are unambiguosly stored in a file labelled metricAndLogsMiddleware_DATE.tar , located in ../results
    %> ./collectMetrcis.sh

This is for the catallactic agents.
For the CNet agents experiments do the same, but using %> ./collectCNetMetrcis.sh
For the ZIP agents experiments do the same, but using %> ./collectZIPMetrcis.sh

In this report the progress of developing the proof-of-concept application in the CATNETS project is presented. Hence the development of the necessary performance measuring components as well as a distributed application to execute on economic-enhanced Grid/P2P platforms and middleware integration are described.