IST-FP6-003769 CATNETS

D4.2

Performance Evaluation

Contractual Date of Delivery to the CEC:	31. August 2006
Actual Date of Delivery to the CEC:	13. October 2006
Author(s):	Oscar Ardaiz, Michele Catalano, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freitag, Liviu Joita, Manuel Medina, Leandro Navarro, Omer F. Rana, Björn Schnitzler, Miguel Valero
Workpackage:	WP4
Est. person months:	19
Security:	public
Nature:	final version
Version:	1.0
Total number of pages:	34

Abstract:

This deliverable describes the work done and the on-going work of WP4 at month 24. A performance measuring infrastructure has been developped for the the prototype and simulator, concering the experiment configuration, data measurement, and data collection. A performance evaluation framework has been prepared to obtain the metrics from the measured data. Initial experiments have been carried out results to test the developped prototype, simulator and the performance measuring infrastructure.

CATNETS Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-FP6-003769. The partners in this project are: LS Wirtschaftsinformatik (BWL VII) / University of Bayreuth (coordinator, Germany), Arquitectura de Computadors / Universitat Politecnica de Catalunya (Spain), Information Management and Systems / University of Karlsruhe (TH) (Germany), Dipartimento di Economia / Università delle merche Ancona (Italy), School of Computer Science and the Welsh eScience Centre / University of Cardiff (United Kingdom), Automated Reasoning Systems Division / ITC-irst Trento (Italy)

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

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

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

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

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

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

Changes

Version	*Date*	*Author*	*Changes*
0.1	06/07/06	FF	
0.2	17/07/06	OA	Paste sections 2.1, 2.2 (UPM) and 3 (UPC,CU)
0.3	20/07/06	OA	Added figures in section 3
0.4	26/07/06	OA	Extended discussion in section 2.4.1
0.5	22/08/06	OA	Added text from UKA on sections 2.2.4 and 4. Several todos included from latest discussions about metrics framework and feasibility of measurements.
0.6	23/08/06	FF	Text added and formating
0.9	30/08/06	OA	Reorganizing section 2.
1.0	31/08/06	FF	Cleaning up
1.1	01/09/06	FF	Another revision, but still not clean, annex from Michele
1.2	15/09/06	FF	English revision of annex

CONTENT

1 Introduction

This deliverable describes the work done and the on-going work in WP4 of in tasks

- T 4.1 Metrics specification and implementation, prototype and simulator (M7-30)
- T 4.2 Evaluation of implemented market mechanisms (M13-30)
- T 4.3 Prototype evaluation (M19-30)
- T 4.4 Performance analysis, comparison, evaluation (M25-30)
- T 4.5 Further research on properties of Catallaxy applied to computer networks (M19-30)

The deliverable refers to the tasks related to performance evaluation of the CATNETS project in year 2. In Table 1 the character of this work is illustrated and set into context. In the first year of the project a theoretical metrics framework was devised as basis for the evaluation of the Catallactic approach. In the second year the implementation of this framework into the prototype and simulator developped in parallel during this year was pursued. First feedback from the practical realization of the framework has been obtained. The third year will focus on the usage of the measurement and performance evaluation infrastructure for evaluation of the Catallactic approach. Additional insight concerning the metrics used will also be obtained and fine tuning of the measurement infrastructure might be required guided by the performance results.

CATNETS PERFORMANCE EVALUATION	
year 1	design of metrics pyramid
year 2	implementation of metrics pyramid in developped prototype and simulator, feedback on chosen metrics, some tests of performance measuring infrastructure
year 3	use of infrastructure for performance evaluation, performance results of Catallactic approach, additional insight and fine tuning of metrics

Table 1. Evolution of performance evaluation work in CATNETS.

The document is divided in four parts: In this chapter the performance measuring goals are recalled. The second chapter describes the implemented performance measuring infratstructure, refering to the metrics to be obtained, the measurement infrastructure within the prototype and simulator, and the performance evaluation framework to obtain performance results from the experiment data. Chapter 3 describes perliminary results on testing of the developped infrastructure. Chapter 4 contains a summary and and outlook to the work to be done in year 3.

1.1 Performance measuring goals

The goal is to evaluate the performance of the Catallactic approach by means of a simulator (see deliverable year 2 of WP2) and a prototype (see deliverable year 2 of WP3).

The simulator targets to compare the Catallactic decentralized approach in two dimensions (Figure 1), one hand with a centralized economic approach and on the other hand with a decentralized non-economic approach. The prototype implements the Catallactic approach only. Certain configurations of experiments of the Catallactic scenario are expected to be tested in both the simulator and the prototype. Qualitatively similar results and tendencies should be observed in both the simulator and prototype for these experiments.

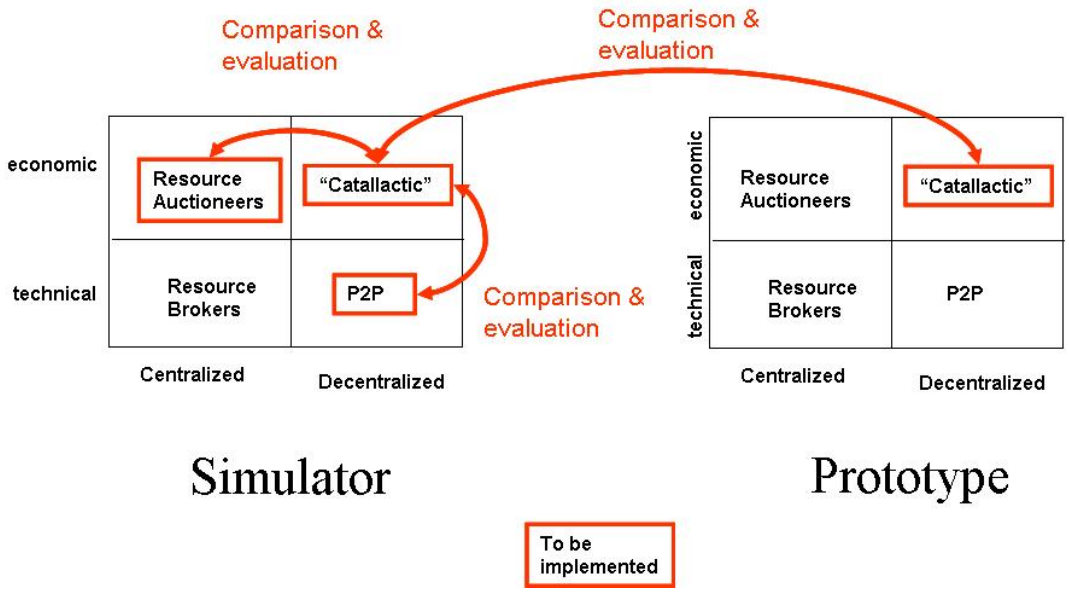

Figure 1. Performance evaluation goals.

2 Performance measuring framework

2.1 Metrics

It is often useful to be able to compare two allocation methods using a single index/number. Such an index provides an aggregated behaviour of an allocation method with reference to a number of features. Figure 2 shows the logical structure of data and indices which has been devised in our approach. As one reaches the upper layers of this pyramid, a loss of information detail results.

In the lower layer of the pyramid, parameters, which are likely to be of significance within an application layer network, have to be selected for the evaluation. These parameters define the raw disaggregated data. Disaggregated indicators provide the first stage of evaluation, and may comprise a number of independently measured values. The raw data could be collected from different experiments into an integrated database.

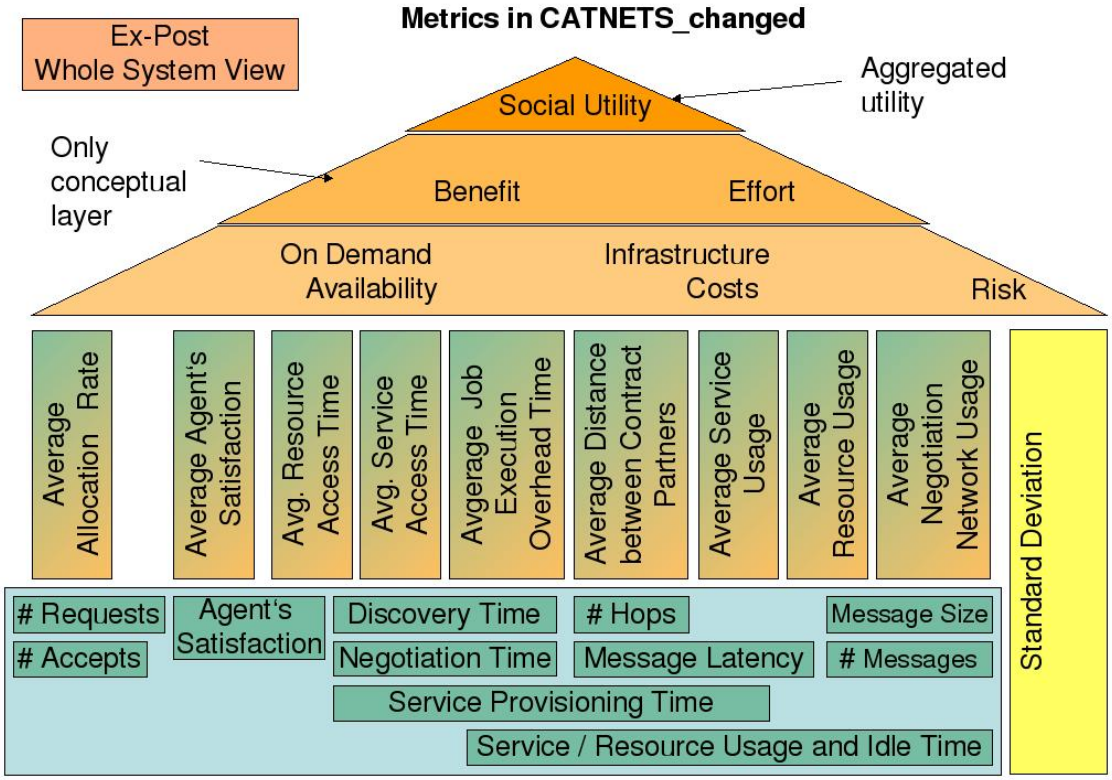

Figure 2. Metrics pyramid (from CATNETS WP4 deliverable of year 1).

For the evaluation with this approach, it is obligatory to take into account a set of characteristics that are not directly comparable, because these characteristics correspond to variables of different dimensions and the unit of measurement. Thus, they have to be made comparable, e.g. by normalization, and then grouped into indicators. An indicator is defined as a ratio (a value on a scale of measurement) derived from a series of observed facts, which can reveal relative changes as a function of time. They also allow the analysis of performance and the predictions of future performance. Finally, the simple and composite indices are computed, which represent benchmarks of performance. They express information in ways that are directly relevant to the decision-making process. Indicators help the assessment, the evaluation, and most important, they help to improve accountability.

The approach is to obtain the technical metrics focuses on providing generic, easily measurable parameters, which can subsequently be aggregated. The upper layer economic metrics will be used in order to evaluate the quality of economic allocation methods. The simple indicator layer defines a set of metrics, which are normalized between zero and one – and are assumed to be independent, which is a classical assumption in statistics. This makes it easier to find functions for the layers above, such as on demand availability and infrastructure cost. The technical metrics may be combined to obtain a framework that enables evaluation of different service oriented architectures.

Technical layer metrics can be classified into: (i) efficiency measures (number of requests, number of accepts); and (ii) utility measures (agent satisfaction). Together, they are a measure of technical benefit which a given service user or provider, represented through a software agent, earns. An additional set is (iii) the set of time metrics (discovery time, negotiation time, service provisioning time), which are measured as the rate of change of market processes.

Message-based metrics (iv) are included in order to measure the activity of users to communicate to find services.

The goal of the evaluation is the social utility index L which is a function of the means and variances of costs and benefits. Integrating the macro-economic models in the metrics framework, the economic policy maker has some preferences about on demand availability and infrastructure costs and is aware of the distribution of the benefits and costs of the agent population.

The metrics pyramid of Figure 1 has been used to guide the implementation of the measurement infrastructure both in the prototype and simulator.

2.2 Design and implementation of the performance measurement infrastructure in the prototype

Performance measuring components for the prototype are important in order evaluate the current proof-of-concept implementation. It is also a research issue to explore the feasibility of including application layer and economic metrics in such an infrastructure.

In the following sub-sections the requirements of the performance measuring infrastructure in the context of the middleware and application components are explained. We address the design of the framework, the used metrics and describe the current implementation.

2.2.1 Requirements on the performance measuring infrastructure

The middleware offers an agent based framework for dynamic location and management of grid services based on economic criteria. It provides mechanisms to locate and manage the registered resources, services and applications, locate other trading agents, engage agents in negotiations, learn and adapt to changing conditions. Furthermore, the middleware offers a set of generic negotiation mechanism, on which specialized strategies and policies can be dynamically plugged in. The service and resource exchange occurs between parties that might join the market in an ad-hoc or opportunistic way.

The requirements, which guided the design of the performance measuring infrastructure, are the followings:

The measurement infrastructure should be able to provide a large number of diverse metrics: the measurement should concern metrics both from the application, the middleware, and the physical level. In addition, there is a need for both technical and economic metrics. The economic metrics provided by the infrastructure should assist the decision makers residing in the users (or applications) with high level metrics on whether to rely on acquiring services and resources when needed, or hosting an infrastructure on their own. Secondly, the measuring infrastructure should allow by means of mainly technical parameters evaluating the infrastructure itself.

The instrumentation by the infrastructure needs to be done at different levels. User agent related parameters should be obtained at the application level. Technical parameters concerning the middleware performance will need to be instrumented at the corresponding levels of the middleware architecture. Finally, the monitorization of the physical resources will be obtained from the base platform. The component of the measuring infrastructure at the

node level should locally gather the values obtained from the components working at the same node but at different layers of the infrastructure.

The data collection should be done centrally, aiming to allow evaluating the prototype at this stage. Nevertheless, it is noted that the software infrastructure should run in a distributed manner on physically different devices and at a later stage with a potentially large number of nodes. For prototype evaluation, however, the number of nodes is small, such that a centralized approach for data collection at this stage appears acceptable.

The evaluation of these metrics, which are collected from the different nodes of the middleware, is done at a central point. The analysis and evaluation of the middleware is done off-line with external tools that provide the needed mathematical functions. The metrics, which the agents need to take decisions, should be processed on-line by the agents themselves.

The development of the performance measuring infrastructure follows the following approach. First, we define the metrics which should be obtained and identify the measurement points within the infrastructure. We proceed with the instrumentation and local data collection. Then, collecting and obtaining the data at a central point is the next step. The evaluation of the data is the final step of the process. In the following sections we describe these steps.

2.2.2 Measured metrics

We assigned metrics to the layers of the software infrastructure. Beginning with the application layer, there are parameters to be measured in the client (which represents the end user) and the application. The client and the application perceive technical parameters, like the *service provision rate*, the ratio between *the number of requests and accepts,* and the *service execution time,* the duration for obtaining an accept. The client as end user will need to transform these technical parameters into economic ones, consider the benefits and the efforts, in order to determine the *utility* obtained by participating in the network. The application might also work with economic parameters, since a business model for this component can potentially be defined.

Related to different levels of the middleware, there are the following technical parameters: the *discovery time* refers to the time the middleware needs to find other agents to negotiate with. The *negotiation time* indicates the duration of the negotiation process. The negotiation process takes place in both the service and the resource market. Each negotiation consists of several messages according to the bargaining strategy. The *message size* is a parameter, which allows describing the communication cost. The *number of messages* is another parameter concerning this cost.

Feasibility of metrics measurements in prototype

We identify which middleware components should provide the metrics defined in the metrics framework pyramid. Some metrics can be measured at the application, others at the economic agents (integrated in the middleware) and others at the base middleware layer. Some metrics does not seem feasible to be obtained in the implemented middleware, but from the base platform layer, i.e. how much CPU does consume each service request.

Metrics that can be measured at the application layer, and also by economic agents are:

- Number of demand requests,
- Number of accepts.

Number of demand requests seen by the application should be equal to the number of demand requests seem by the economic agents. However, if we consider that there is a CAP (Catallactic Access Point, see WP3 deliverable of year 2) between both entities, there might be differences between both measures.

Metrics that can be measured by economic (Catallactic) agents are:

- Number of requests and number of accepts
- Number of negotiation requests and negotiations accepts
- Agents' satisfaction
- Discovery time
- Negotiation time
- Number of messages

Discovery time should be measured by the Catallactic agents, since the agents have full control of the negotiation protocol, and can determine when the discovery effectively finished (the agents do not use the middleware's provided search mechanism).

Negotiation time should be measured by the Catallactic agents: the agents are in control of the negotiation protocol and can determine when the negotiation has finished (agents are not using the economic mechanisms framework provided by the middleware. which would handle the negotiation protocol and provide the measurement of related metrics).

Number of messages should be measured by the Catallactic agents, and only the messages related to a negotiation will be counted. This information is available only to the agents, as the middleware cannot distinguish a transaction from another.

There are a number of other metrics, for which it has not been finally decided where and how they can be best measured: service execution time, resource usage time, hops, latency, message size, and network transfer.

Service execution time: It is important to notice that in the proposed architecture the actual service invocation is not handled by the BasicService agent, but from the application level, outside of the control of the middleware (which only participates in the allocation process). It is possible to measure the service execution time as seen from the application clients.

Resource usage time: It is not clear how to relate the resource usage to one specific transaction. For instance, if the service is a web service, which runs in an application server, it is hard to measure the CPU consumption for each service request. But it is possible to measure Average Resource Utilization, which is the metric intended to be composed from resource provisioning time; this metric is reported for the CPU resource by using Linux operating systems calls.

Hops: In the prototype we do not have direct control of the P2P overlay network, since it is handled by JXTA. Currently, it is not possible to know how many hops a message follows.

Latency time: We do not have a way to measure JXTA message delays. We might implement some latency measurement service in the middleware, based on some exiting tool, if considered appropriate.

Message size: We do not have a way to meter JXTA message size.

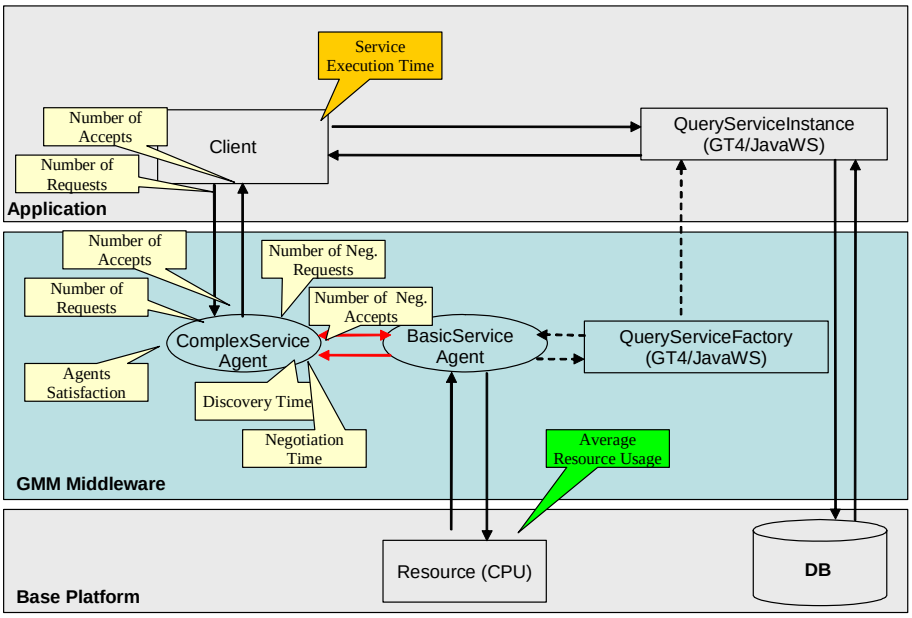

Figure 3. Measurement of metrics in the prototype test experiments

2.2.3 Measurement infrastructure

Instrumentation and local data collector

In our approach we took the design decision that data from one node should be locally collected. This way, we obtain at each node an event trace, which includes the metrics from the different middleware layers. Provision of these metrics is through agents (Figure 4). The event trace contains the time stamps of the events, the metrics itself and a number of attributes like the agent number, transaction number, and others, in order to allow a detailed analysis of the behavior. The local data collector manages this data structure. In terms of implementation, a circular structure is used such that its size is controlled.

Access to this data structure is given in two ways. One hand, the data can be written to a file (log file), and on the other hand the local data collector can send it regularly to a global metrics collector located on a particular node.

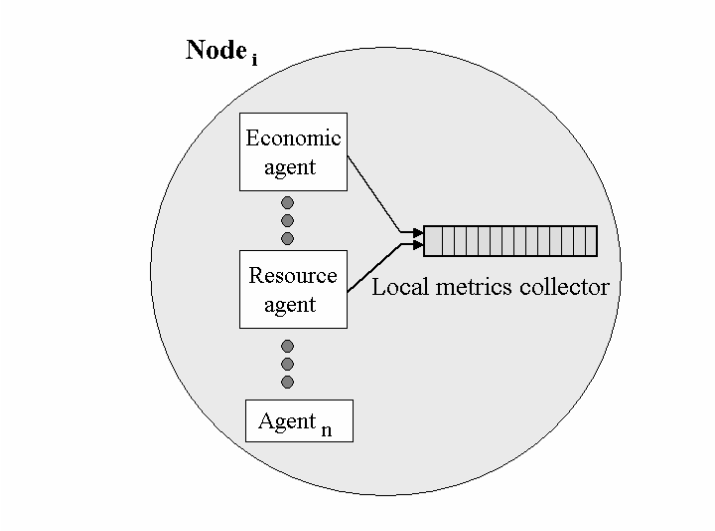

Figure 4. Local metrics collector.

Global metrics collector

The data obtained at the different nodes is send to the global metrics collector, which resides on a particular node of the system. Data is send by means of a push mode: local data collectors initiate the sending to the global collector according to a configured behavior. The global metrics collector then processes and organizes the data into a format suitable for external packages. For initial testing, the data has been analyzed using standard software packages. For intensive performance evaluation the performance evaluation framework has been developped.

Issues

In the performance measuring infrastructure of the prototype, the first challenge was to obtain metrics from all layers of the system. As such, this framework needs to work with and go beyond other monitorization toolkits which exist for distributed systems, but which mainly focus on the physical resources. For our purpose we also need to include application and middleware data. These higher level metrics are necessary to be considered in our application context, since they allow extracting metrics, which can be interpreted in economic terms.

The second challenge was that there are different destinations for some of these metrics. These metrics, in addition, need to be conveyed to different destinations: On one hand there is a central metrics collection point, to which most of the data is sent (except some data of the client) and where the system is analyzed and evaluated. On the other hand, there are the participants (the applications) as destination of metrics, since they need application layer metrics in order to take decisions and evaluate their performance. One possible solution, to route data to particular groups, has not yet been implemented in our framework. Our view on this is to apply publish/subscribe mechanisms in order to assign groups to metrics.

Another issues, which may get important at a later stage of the project, are clock synchronization and scalability. For larger scale usage beyond the current experimental settings, the automatization of the clock synchronization between the nodes and the global metrics collector is an issue which needs to be solved. The scalability of the performance evaluation framework could also become critical for certain experiment configurations. Once the system gets deployed in a larger scale beyond controlled conditions, this scalability

problem may affect the number of parameters which can be monitored, and may require additional measures to tackle the size of the traces obtained.

2.3 Design and implementation of the performance measurement infrastructure in the simulator

The measurement infrastructure in the simulator is used to measure a set of predefined economical and technical metrics. The term *measurement infrastructure* is understood as a generic way to collect and measure different kinds of metrics and to store them during the simulation process independent of their type.

In the following, different aspects that are used to realize such an infrastructure are discussed: Section 2.4.1 describes the measured metrics for the central and Catallactic cases. Section 2.4.2 introduces the technical concepts of the measurement infrastructure.

2.3.1 Measured metrics

Deliverable D4.1 describes a set of technical and economical metrics for the evaluation of the central and Calallactic allocation mechanisms. The challenge we encountered is that some of those metrics can be measured by the simulator but not by the middleware and vice versa. Furthermore, we identified some metrics that can be measured in the Catallactic case but are fixed in the central case. An example for such a metric is the service discovery time: In the Catallactic case, several nodes need to be contacted in order to find adequate counterparts for a service provisioning. In the central case this time is fixed, as the “discovery” of relevant services is realized by a central component, i.e. the auctioneer.

For the first step of the simulator implementation, a partial set of the envisioned metrics framework is implemented. In the following, an overview of the measured metrics is given. The implementation of the metrics is independent from the economic model.

Basic Service Provisioning Time

Definition: The basic service provisioning time represents the time that is required to execute a basic service. This includes the allocation time for the basic service and the allocation time for the resources needed for basic service execution. In the centralised case, allocation time is the time needed by the central auctioneers to allocate the basic service and the related resource bundle. In the Catallactic case, allocation time is the time needed to for basic service and resource discovery and bargaining.

Measurement: In terms of code, basic service provisioning time is the time that is required for a single iteration in the following "for" statement in class `ComplexServiceAgent`:

```
for (String bsName = accessPatternGenerator.getNextBS(); bsName !=
null; bsName = accessPatternGenerator.getNextBS()) {
      .....
} // for each BS in CS
```

This statement iterates over all the Basic Service which compose a Complex Service. For time calculation, the method `GridTime.getTimeMillis()` is used. It returns the time in milliseconds from the beginning of the simulation. The Basic Service Provisioning time is the

difference between the value returned by `getTimeMillis()` at the end of the iteration and the value returned at the beginning.

Complex Service Provisioning Time

Definition: The complex service provisioning time is the time that is required for the (complete) execution of a Complex Service and is the sum of the provisioning times of the component basic services. If some Basic Service in the Complex Service fails, then Complex Service Provisioning Time is not calculated for that Complex Service.

Measurement: In terms of code, it is the time needed for all iterations in the "for" statement of

```
for (String bsName = accessPatternGenerator.getNextBS(); bsName !=
null; bsName = accessPatternGenerator.getNextBS()) {
      .....
} // for each BS in CS
```

This time is the difference between the values returned by `getTimeMillis()` before and after the execution of the “for” statement above.

Complex Service Agent Allocation Rate

Definition: The complex service agent allocation rate is the ratio between the number of successful requests for complex services and the total number of requests for complex services.

Measurement: In terms of code, this metric is measured at the end of `run()` method of class ComplexServiceAgent. A counter C1 for successful requests is increased at the end of the previously mentioned "for" statement if all BSs have been successfully allocated. A counter C2 of total request is increased if CS != null in the run() method. The Complex Service Agent Allocation Rate is the ratio between C1 and C2.

Application Allocation Rate

Definition: The application allocation rate is the weighted average of all (Complex Service Agent) allocation rates and is calculated at the end of simulation.

Measurement: In terms of code, this metric can be measured outside the simulator code. For example, for every ComplexServiceAgent we could record at the end of the run() method the values of the counters previously mentioned. This way, the weighted average can be easily calculated.

2.3.2 Measurement infrastructure

A central metrics logger realizes the logging of metrics. Basically, the logger is represented by a singleton class called `MetricsLogger`[1] in the simulator. The class can be accessed by

[1] The class can be found in the `org.catnets.optorsim.utils` package of OptorSim.

every other class that measures any metric. An overview of the methods provided by the class is outlined in Table 6.

Each time, a new metric measurement is reported to this class, the attribute and value of the metric as well as further information concerning this metric are stored in a CSV text file. The use of text files as an output media is used due to simplicity. In case we encounter scalability problems due to the use of text files, the output medium can be easily switched to a database. For a further evaluation of the stored metrics, the output text files can be easily imported into a database.

Table 2. Methods of the Metrics Logger class

MetricsLogger.class: Method Summary	
`void`	`close()`
`static MetricsLogger`	`instance()`
`boolean`	`isLogging()`
`void`	`log(long timeStamp, org.catnets.optorsim.infrastructure.AlnSite site, org.catnets.optorsim.negotiations.Negotiator negotiator, java.lang.String name, double value)`
`void`	`setLogging(boolean doLogging)`

The `MetricsLogger` class provides a function called "`log`" which is called to store a particular metric. For instance, an instance of a `BasicServiceAgent` class may call this method to store the time that is required for an allocation. Beside the name and the value of a metric, the method stores further information such as the time of measurement as well as the site and the negotiator who measured the particular metric. For a detailed overview of the different classes and their meaning in the simulator, the reader is referred to deliverable WP2 year 2.

2.4 Evaluation framework design and implementation

This section is devoted to the description of the scripts, which implement the evaluation process of the CATNETS project. In the following section, we will show how the evaluation process should be adapted to the simulator and prototype environments.

2.4.1 Overview

The evaluation process is composed by four steps (Figure 5).

- Collection of input from the simulator and prototype functions storing it in a database
- Economic metric evaluation by an application which perform the application of formulas automatically communicating with the database
- Optional selection of data in order to perform analysis on a sub set of data
- Store the results in a single database in order to map the parameter grid with results.

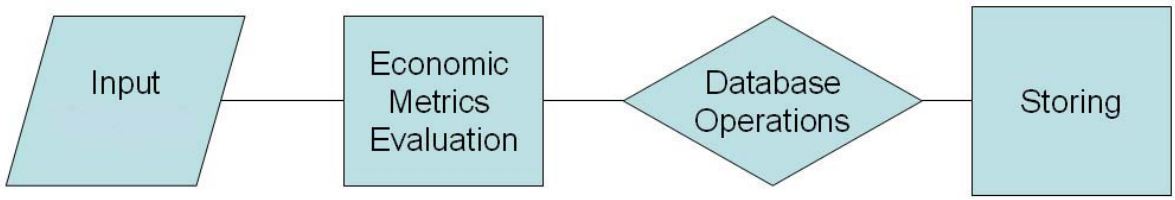

Figure 5. Evaluation process.

Input data Collection

The evaluation process requires the analysis of a large amount of data. The simulator and prototype/middleware/application tests have been organized in several experiments, each of them characterized by the parameter settings and scenarios. In order to perform a systematic investigation on the dataset, it needs a:

- definition of the store data structure
- selection of a database standard platform
- automatization of metrics evaluation over the different experiments
- storing of final economic metrics

The best way to map the general evaluation process overview in an efficient process is to build an application communicating with the database taking as inputs the database records consisting of technical metrics output of simulator and prototype, and providing as output the economic metrics evaluation.

2.4.2 Database use

The design of the metrics evaluation begins with the selection of the metrics and parameters for the description of the Catallactic/centralized scenario. The metrics shown in section 2.1 are collected in a database and fill a group of tables according. Each table is filled by different functions, which are called in different times, i.e. runtime and at the end of the experiments.

The basic tables are:

Experiment table: collects the main information about the parametrization, the time occurred to run simulation/experiment and the number of agents. Each experiment is labelled with an id.

Transaction Table: collects the main technical metrics regarding each transaction occurred between Complex services and basic services.

Usage table: is filled by the main technical metrics related to the usage of the market from agents.

Distance table: includes the main metrics regarding the distance of the exchanging agents.

Metrics table: is filled by the intermediate and final metric layers defined in the pyramid (see section 2.1).

Each record of the table is the result of processing the technical metrics data and it refers to each experiment.

Transaction Table

After each successful contract between a complex service and a basic service (including successful sub-contracts with several local resource managers), the complex service stores the transaction related data to the metrics database (transaction level) in the transaction table (Table 3). The transaction table is written by the Complex Service at runtime.

Table 3. Transaction table.

Column	Description	Unity of Measurement
exp id	The id of the experiment, to distinguish between several experiments within one table (foreign key for experiment table).	Integer
complex service id.	The unique number of the complex service	Integer
basic service id.	The unique number of the basic service	Integer
1st resource id.	The unique number of the first sub-contracted resource	Integer
2nd resource id.	The unique number of the second sub-contracted resource	Integer
3rd resource id.	The unique number of the third sub-contracted resource	Integer
Timestamp	Time elapsed from the begin of simulation and experiment (Simulation time reference/Realtime for prototype).	Milliseconds
Number of Demand Requests	This metric counts the number of launched discovery processes until this contract is achieved.	Integer
Number of Negotiation Requests	This metric counts the number of launched negotiation processes until this contract is achieved	Integer
Agent Satisfaction	Still needs to be defined for centralized approach. In the decentralized case, it weighs the service/resource quality and the price, i.e. all possible basic service are ranked at complex service level depending on its negotiation start prices and their self-indicated quality (e.g. average response time) in comparison to the desired objectives given in the bill of services (measured in %). After the contract could be achieved, these values are compared again to the desired objectives and this value contributes to the satisfaction.	Real
Discovery Time	This metric is used to measure the time to find a given set of possible negotiation partners.	Milliseconds
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	Milliseconds
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.).	Milliseconds
Resource Provisioning Time (Effective Job Execution Time)	The evaluation framework defines the resource provisioning time as the resource usage time of one transaction. (This metric is only taken in to account for the prototype, as provision time cannot be fixed.)	Milliseconds
Job Execution Overhead Time	Total Job Execution Time - (Discovery Time + Negotiation Time + Provision Time)	Milliseconds
Total Job Execution Time	The total job execution time is defined as a sum of discovery time, negotiation time (waiting time in centralized approach), network transfer time and provisioning time (which is - however - fixed for simulation).	Milliseconds

Experimentation table

At simulation/prototype run initiation an introductory entry is stored to the experimentation table (Table 4.). This information should enable a distinct identification of the experiment. It is written by the configuration scripts at start-up.

Table 4. Experimentation table.

Column	Description	Unity of Measurement
exp id	The id of the experiment, to distinguish between several experiments within one table (foreign key for experiment table).	Integer
description	Contains data about which density/dynamicity is used	Text
approach	centralized/decentralized/prototype	C/D/P

start timestamp.	Experiment start time (Realtime)	dd.mm.yyyy hh:mm
end timestamp	Experiment stop time (Realtime).	dd.mm.yyyy hh:mm
Agents	Total number of agents CS+BS+RS for each Experiment.	Integer

Usage Table

When simulation is finished, the usage times of each agent are stored to the usage table (Table 5). The usage table is written by each agent after simulation.

Table 5. Usage table.

Column	Description	Unity of Measurement
exp id	The id of the experiment, to distinguish between several experiments within one table (primary key for transaction table).	Integer
agent id	The unique id of the agent.	Integer
Service Usage	The service usage is evaluated by the ratio between the service provisioning time and the total simulation time (only for simulator).	Milliseconds
Resource Usage	The resource usage is evaluated by the ratio between the resource provisioning individual time and the total simulation time (only for simulator).	Milliseconds
Number of Messages	This value counts the number of messages.	Integer

Distance table

Table 6 helps to calculate the distance between contract partners. This calculation is sufficient to be done after the simulation. It has to be understood as an adjacency matrix, showing hops and latency times between communication partners. This enables to store the numbers at configuration time of the simulation, disburdening the calculation. Distances and times is considered to be the same on the way back, so every pair is itemized only once.

Table 6. Distance table.

Column	Description	Unity of Measurement
exp id	The id of the experiment, to distinguish between several experiments within one table (primary key for transaction table).	Integer
sender agent	Message transmitter	Integer
receiver agent	Message receiver	Integer
hops	Distance between the partners in hops	Integer
latency time	Distance between the partners in latency time.	Milliseconds

Metrics level

Now, the MATLAB scripts are called. These must generate the data according to the metrics pyramid. The data will be stored in the following metrics level table (Table 7).

Table 7. Metrics level table.

Column	Description	Unity of Measurement
exp id	The id of the experiment, to distinguish between several experiments within one table (foreign key for metrics table).	Integer
allocation rate	Average Allocation Rate	Normalized [0;1]
agent satisfaction	Average Agents' Satisfaction	Normalized [0;1]
service access	Average Service Access Time	Normalized [0;1]
resource	access Average Job Execution Overhead Time	Normalized [0;1]
distance	Average Distance between Contract Partners	Normalized [0;1]
service usage	Service Usage	Normalized [0;1]
resource usage	Resource Usage	Normalized [0;1]
network usage	Network Usage	Normalized [0;1]
availability	Availability	Normalized [0;1]
infrastructure costs	Infrastructure Costs	Normalized [0;1]

risk	Risk	Normalized [0;1]
utility	Utility	Normalized [0;1]

The scripts enable the user to select the data of each experiment from the input tables and then evaluate the metrics. Furthermore, it saves the results on the database as the schema represented in the previous table.

2.5 Discussion

The performance measurement infrastructure to obtain the metrics pyramid (see section 2.1) has been realized as a real implementation. This implementation consists of two main components, the measurement infrastructure and the performance evaluation framework. The measurement infrastructure has been integrated into the prototype and simulator, respectively, and obtains raw data. The performance evaluation framework has been realized with MATLAB and database integration and is called after simulation to compute the performance numbers, which should assess the Catallactic approach.

In the practical development of the performance measurement infrastructure it was found that some of the proposed metrics are difficult to obtain in the prototype, like the number of hops and latency, due to the selected tools and mechanisms with which the prototype has been realized. We also noticed that in the simulator a few metrics are only meaningful in a particular scenario, like discovery time in the decentralized case, while in the centralized case the discovery function is not part of the mechanism.

Experiments with the developed simulator and prototype are need in order to fine tune in another iteration the performance measurement infrastructure. The experiments will both provide the performance results for the Catallactic mechanism but also provide feedback on which metrics form the devised pyramid will allow us need particular focus for this evaluation.

3 Performance evaluation tests with the prototype

Given the developed prototype (see deliverable WP3 of year 2) and the implemented measurement infrastructure and evaluation framework, we have carried out test runs to apply all these components togehter. These test runs had the main purpose to assure that the components work correctly togehter and are the step just prior to the experiments which should lead to obtain the performance numbers – given the performance measurment infrastrucure.

3.1 Experiment set up

We have carried out experiments with the current version of the middleware, allowing a preliminary performance assessment of the Grid Market Middleware (GMM).

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

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

Services start selling the resources at a price, which is solely influenced by the node's utilization, following the pricing model presented in year 2's deliverable of WP3. Services get involved in negotiations and the price will also be influenced by demand. If a Service agent is selling its resources, it will increase the price to test to what extend the market is willing to pay. When it no longer sells, it will lower the price until it becomes competitive again or it reaches a minimum price defined by the current utilization of the resource.

In order to test the performance of the market based resource allocation mechanism, we setup controlled experiments deploying several instances of the middleware in a Linux server farm. Each node has 2 CPU Intel PIII 1 GHz and 512 MB of memory. The nodes in the server farm are connected by an internal Ethernet network at 100Mps.

We deploy GMM and the Web Services on six nodes (named arvei-7 to arvei-12). On each node we also deploy a Web Service, which performs a CPU intensive calculation on the machines, increasing load. These Web Services are exposed in a Tomcat server. Access to execute these Web Services on the Resources is what will be negotiated by the services and the clients.

The experiments consist in launching 2 clients concurrently from 2 other nodes, which are not running the Web Services. Each client performs 50 requests, in intervals of 10 seconds. Whenever a client wins a bid with a service, it invokes the Web Service in the selected node. The complete experiment runs for about 10 minutes. We generate a baseline load on three nodes (arvei-10, arvei-.11 and arvei-23) of 25% of CPU usage to simulate some background activity, how is generated is explained in deliverable 3.2. Also to better test autonomic load balancing, we artificially stressed one of the nodes (arvei-10) up to 95-100% of CPU usage for a short time during the experiment.

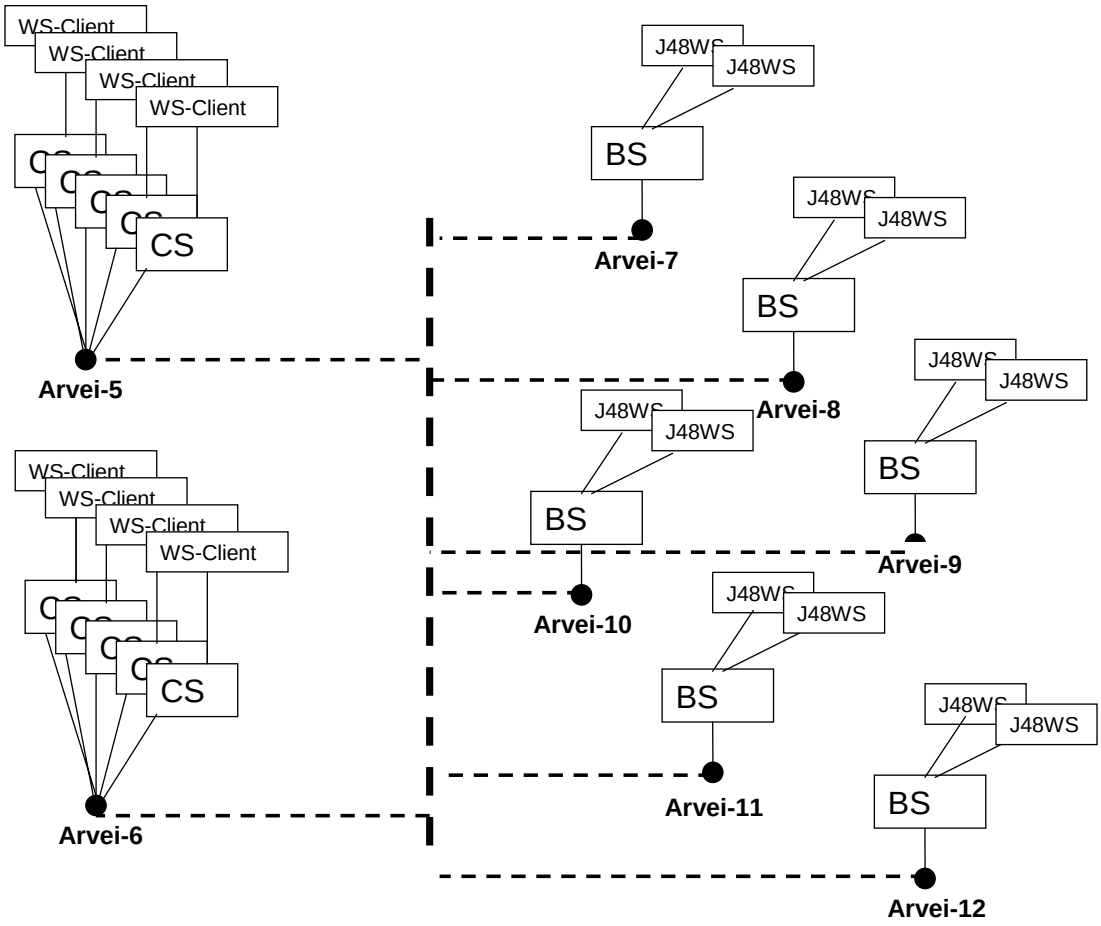

Figure 6. Experimental setup.

3.2 Results and discussion

The data obtained with the performance evaluation infrastructure in the experiment has been passed to the evaluation framework, which implements the metrics pyramid layers. The purpose is to carry out a test of the complete components developed for the performance evaluation.

The data used as inputs for the evaluation is the following:

1. *allocation*: an entry by each successful negotiation for a Basic Service, reported by the Complex Service
2. *price*: a periodic report of the price done by Basic Services
3. *utilization*: a periodic report of the CPU utilization done by the Resource agents, as a fraction (0.0 to 1.0)
4. *negotiation.time*: time needed to negotiate for a Basic Service, reported by the Complex Service (in milliseconds)
5. *execution.time*: time needed to actually execute the service, reported by the Complex Service (in milliseconds)

Each metric follows the following format:

Timestamp, a long with the time stamp of the metric generation;
Node where the metric was issued;
Agent that generated the metric;
Name of the metric;
Value of the metric;

Table 8. Adaption.

Available data	Referred metric
price	*satisfaction*
execution.time	*provisioning.time*
negotiation	*negotiation.time*
utilization	*resource.usage*

The data used is mixed in nature because it is collected periodically (*price* and *utilization*) or after each successful transaction (*execution.time*, *negotiation.time* and *allocation*). In section 2.2, the major evaluation process requirements and those met by the actual dataset are shown.

In year one's deliverable of WP4 the features that data should have are shown. Essentially, each data for each technical metric is referring to a single transaction and it has to be collected by the agent interested in it (CS, BS, RS). In this respect, *price* and *utilization* data are collected periodically and independently by each transaction, giving a number of observation respectively of *943* and *856*, while *execution.time*, *negotiation.time*, and *allocation* are *100*-observations dataset and are collected each of them by the CSs.

As can bee seen in Table 8 much of the input data, as established in the metric pyramid, is not available and for the data considered here there was some difficulties to adapt it to the original schema. In the table the adaptation of data with respect the original schema is shown. For what price concerns (satisfaction mapping), the quality data that define the satisfaction ratio is missing (see deliverable WP4 Y1 for the context of this metric). It has been supposed that quality, measured as average response time, is equal for each transaction and of 30 milliseconds. Furthermore, as a correspondence between agents misses, in order to get the price corresponding each transaction, the closer *"execution"* observation has been collected. To achieve this goal, the time stamp of *execution* and *price* has been matched. The computation works as follow: the first price data that has been collected after the execution data time stamp has been considered. Repeating the above procedure for each execution time, a *100*-observation data set both for *price* and *utilization* it has been extracted. The following step is to calculate the first aggregation level, which contemplates the construction of a normalized metrics set. In particular, the *access.time* metric as the sum of *execution* and *negotiation.time* is calculated.

To normalize the sum it has been applied the following metric:

$$access.time = e^{-\beta(execution.time+negotiation.time)}$$

where $\beta = 0.00005$ is a parameter choose arbitrarily.
Once obtained the final data set, it is possible to evaluate the economic metrics: satisfaction, *access.time* and *resource.usage*. Finally, *On.Demand.availability (ODM)* and *Infrastructure.costs (IC)* are calculated as follow

$ODM = access.time + satisfaction$

$IC = resource.usage$

and, applying the mean and standard deviation of both the metrics on the right side of equation, as in deliverable WP4 Y1, we obtain the inputs for the final index, where the results are shown in Table 9.

$$L = \alpha\left[1 - 1/2\left(\mu_{access.time} + \mu_{agent.sat}\right)\right]^2 + \alpha(1/2)^2\left(\sigma^2_{agent.sat} + \sigma^2_{access.time}\right) + \beta\left(\mu_{resource.usage}\right)^2 + \beta_{resource.usage}$$

Table 9. Results.

Metric	***Value***
L	*0.43508*
$\sigma_{resource.usage}$	*0.32025*
$\mu_{resource.usage}$	*0.48686*
$\sigma_{access.time}$	*0.013218*
$\sigma_{agent.sat}$	*0.015615*
$\mu_{agent.sat}$	*0.46295*
$\mu_{access.time}$	*0.080522*
α	*0.5*
β	*0.5*

The metric (mean) values in Table 9 close to 0 means that the system for the related metric works well while the metric (st.deviation) values close to zero means that the dispersion around the mean value is low and the again the system works well. However, one experiment is not sufficient to say more. The metrics framework can be confirmed to work well when it has been tested in many experiment, and when many different parameter settings have been compared.

4 Summary and future work

The developped performance measuresment infrastructure has built on the metrics pyramid devised from a theoretical point of view during the first year of the proyect.

In this second year, in parallel to the ongoing development of the simulator and prototype, a performance measurement infrastructure has been developped. This infrastructure counts with tools to faciliate the configuration of experiments, components to measure and components to provide raw data.

In order to transform the raw data into aggregate values, a evaluation framework which uses databases and relies on MATLAB routines has been implemented. This framework is expected to ease and accelerate the process of obtaining performance results form the experiments in a fairly automatic way.

Initial experiments where the performance measuring infrastructure has been used together with the evalution framework have been carried out. This step focussed on the testing of the

deveolpped components, verify if they work together correctly, and is prior to the obtention of performance numbers.

The practical developement and implementation of components has lead to new insight into the theoretically devised year one's performance pyramid. The feasibility of some of the proposed metrics has been confirmed, but it was also found that others are not easily obtainable in all scenarios or limited to the simulator or prototype only.

The stabilization of the developped simulator and prototype in year 3 should allow obtaining performance numbers for the Catallactic mechanisms. This task will also allow to fine tune the performance measuring metrics and the developped infrasture.

5 References

[ACC+05] 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 within AAMAS,* Utrecht, Netherland, July 2005.

[Diet05] Diet Agents Platform, December 2005, http://diet-agents.sourceforge.net/

[ERA+03] T. Eymann, M. Reinickke, O. Ardaiz, P. Artigas, F. Freitag, L. Navarro, "Self-organizing resource allocation for autonomic network", *Proceedings. 14th International Workshop on Database and Expert Systems Applications*, Germany, 656- 660, Prague, Chech Republic, 2003.

[FLC02] F. Freitag, J. Caubet, J. Labarta, "On the Scalablitiy of Tracing Mechanisms". *Euro-Par,* Paderborn, Germany, August 2002.

[Glob05] Globus Toolkit, December 2005, http://www.globus.org/

[JRA+05]L Joita, O Rana, O Ardaiz, P Chacin, I Chao, F Freitag, L Navarro, "Application Deployment using Catallactic Grid Middleware", *3rd International Workshop on Middleware for Grid Computing,* Grenoble, France, November 2005.

[Jxta05] Project JXTA, December 2005, http://www.jxta.org/

[LHF04] K. Lai, B. A. Huberman, and L. Fine, "Tycoon: A Distributed Market-based Resource Allocation System," HP Lab, Palo Alto, Technical Report cs.DC/0404013, Apr. 2004

[MCC04] Matthew L. Massie, Brent N. Chun, and David E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience". *Parallel Computing,* Vol. 30, Issue 7, July 2004.

[Plan06] PlanetLab. http://www.planet-lab.org/

6 ANNEX I

Documentation of the MATLAB GUI for CATNETS Evaluation Process

1 Introduction

This is documentation for Matlab scripts involving a first attempt to develop a platform corresponding to the evaluation process of the CATNETS project. In the future, the data of the simulator and prototype will be inputs of the same scripts. In this documentation the following goals are addressed:

• produce artificial technical metrics
• evaluate intermediate indicators
• calculate aggregated index
• write into the technical metrics a database
• read the technical metrics as tables from a database
• manage Sql queries on the technical metrics

In the following document, the above issues are addressed and the intermediate and final outputs regarding the procedures/scripts implemented in Matlab are shown.

1.1 General overview

The evaluation process is visualized in Figure 5 of section 'Evaluation framework design and implementation'. The first step is to get data as inputs, like artificial technical metrics or actual metrics. The second step is to evaluate the economic metrics layer and the higher level metrics. The third step is to perform a selection with some criteria of the economic metrics. The process finishes with storing the data selected in a single database.

To achieve the functions comprised in the above steps it is convenient to automate the whole process. The best way is to use an application working with a database. For this purposes two scripts called *GenTables*, which perform an artificial generation of technical metrics, and *CatnetsValue*, which perform the evaluation process, are available. Both scripts are built upon the database and the GUI (graphical user interface) toolboxes available for Matlab, in order to manage easy and automatically the evaluation process. *GenTables* script accept as inputs the number of agents (Complex Services, Basic Services and Resources), and provide as output the data in the same schema as tables 2-3-4-5-6 in section 'Evaluation framework design and implementation', storing it in a database using a given ODBC or JDBC connection. The behaviour of *CatnetsValue* is more complex because it reads the technical metrics data stored in the tables provided by *GenTables* and it creates a final table and stores the economic metrics following the general pyramid layer schema (see Figure 2 of section 2).

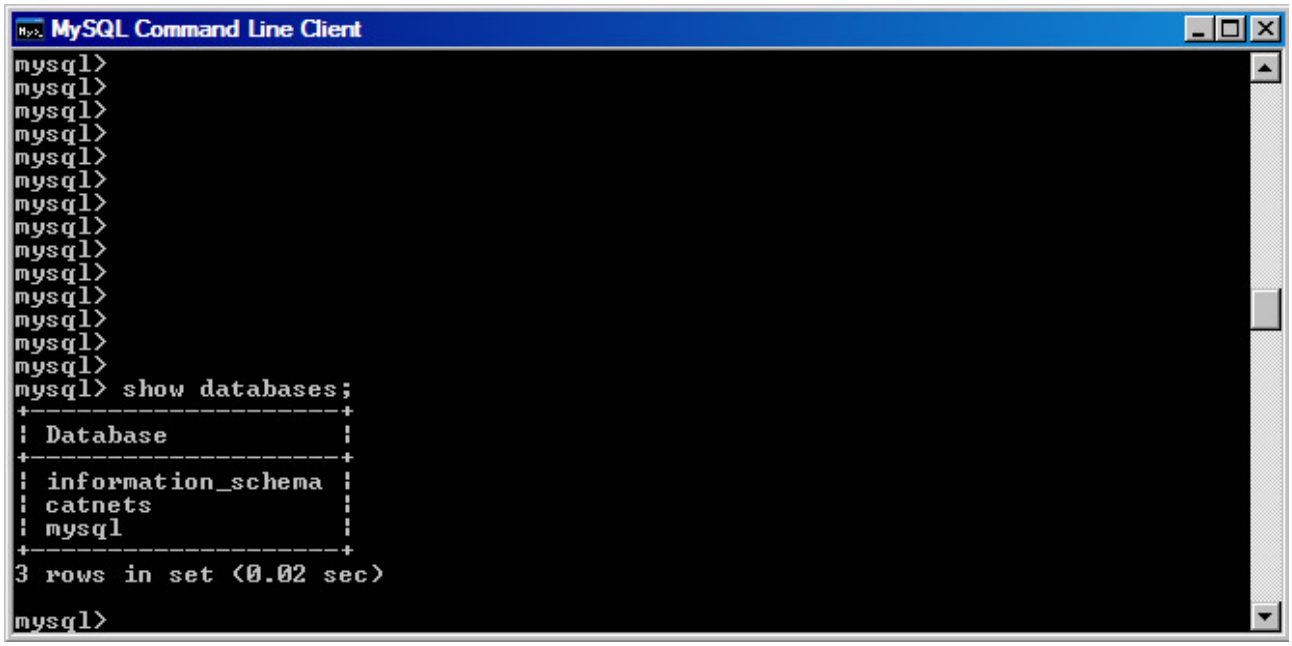

Figure A.1: databases available in MySql.

2 How to set database connection for ODBC MySql platform

The Database Toolbox, and therefore *GenTables* and *CatnetsValue,* supports the import and export of data from any ODBC/JDBC-compliant database management system. The database system used is MySql. So in order to follow the instruction below MySql 5.0 version or above is needed.

In order to perform a SQL operation on data, the setting of a database connection is needed. At the moment, the scripts are set for being connected to the database 'catnets'. The general command in Matlab to perform the connection is

conn=database('databaseSourceName','username','password');

so the actual command in the code is

conn=database('catnets','','');

Furthermore, to perform the connection a database connection for the MySql database with database source name "catnets"[2] is established.

Once the database connection has been created, the next step is to create a new database called 'catnets'. Start the MySql command line client and type

create database catnets;

to see the result type the command "show databases;" and the list of available databases should state as in Figure A.1.

[2] To set the connection follow the instructions in the official reference manual available at http://downloads.mysql.com/docs/refman-5.0-en.a4.pdf#search=%22refman-5.0-en.a4.pdf%22

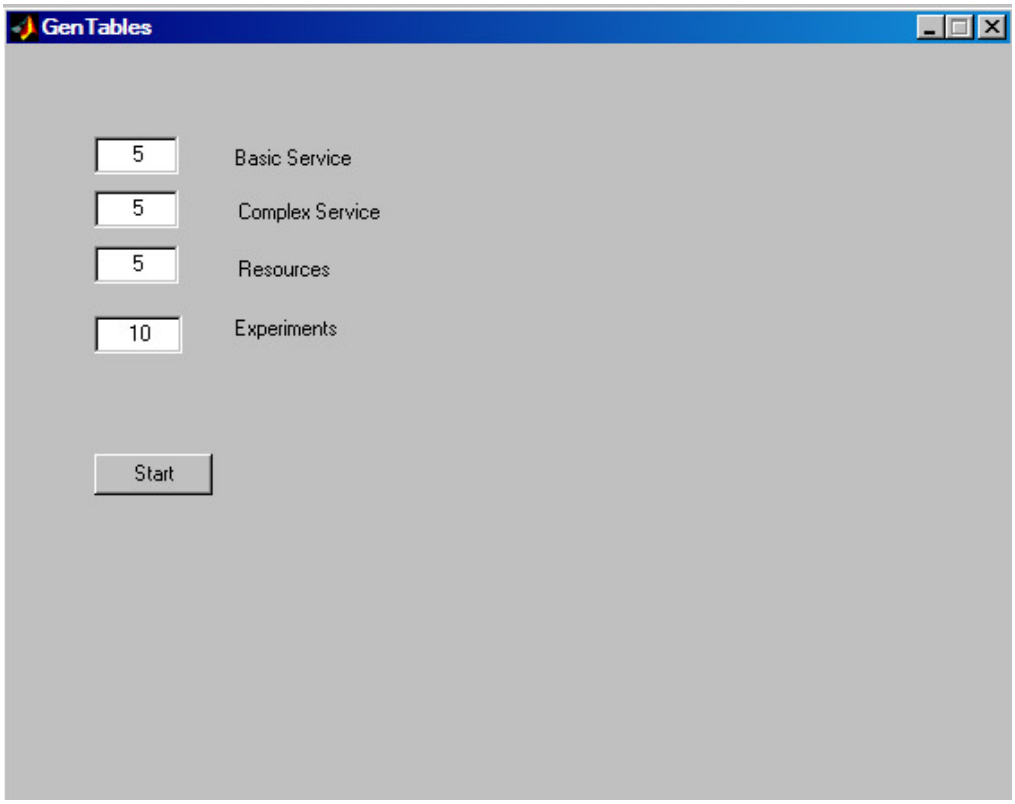

Figure A.2: GenTables GUI

2 Install and run GUI

Once the 'empty' database is ready, the scripts can be used. Save the package into a directory; select in Matlab as “current directory” the directory where the package is installed, and type ’gentables’ on the Matlab command window. Now, a GUI as in Figure A.2 should open and you should be ready to work.

3 Artificial technical metrics

In order to grant the future evaluation process for CATNETS project, we want to develop a procedure in Matlab, which is robust, corresponding to the methodological approach developed into deliverables and preliminary discussions. As we have not yet a set of data coming from the simulator and prototype experiments, we need to simulate the evaluation process starting with artificial data. Figure A.2 shows the mask enabling the user to input the number of agents (complex service, basic service and resources) and the number of the experiment. Each experiment is run with the agent number set and comprises the same set of artificial technical metrics. When the scripts run, no topology and interaction of agents is taken into account. Each technical metrics is obtained drawing a random number from a uniform distribution with arbitrarily support. For example, the “Number of Demand Requests” data is obtained setting a support equal to 20; meaning that the maximum value for each agent of launched discovery processes until the contract is achieved is at maximum set to be 20. The scripts extract randomly the metrics with respect to the unity of measurement (i.e. integer, real or time format) and the interval of definition (i.e. comprised between [0,1]).

Figure A.2 shows an example: Set the input as in Figure A.2, and set the number of complex services, basic services and resources equal to 5 and the number of experiment equal to 10. Then click on the Start button. Now, for each agent the scripts draw a random number and if it's lower than a certain probability *p,* then the agents perform a transaction and related artificial technical metrics are produced

and stored in the tables. Then the metrics are imported and written into the database '*catnets'* with the following tables:

• Transactions Table, corresponding to Table 2
• Experiment Table, corresponding to Table 3
• Usage Table, corresponding to Table 4
• Distance Table, corresponding to Table 5

To see the list of tables created type on the MySql command line client (Figure A.3):

show tables;

and to see the content of table experiment type (Figure A.4):

select * from experiment;

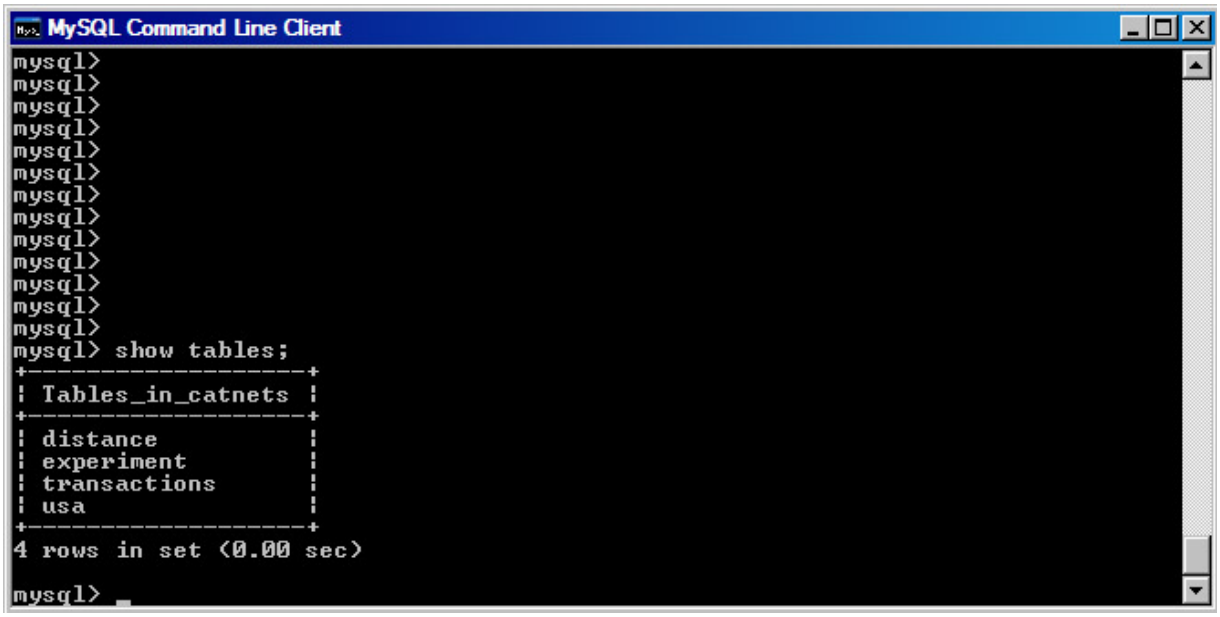

Figure A.3: Tables storing artificial technical metrics

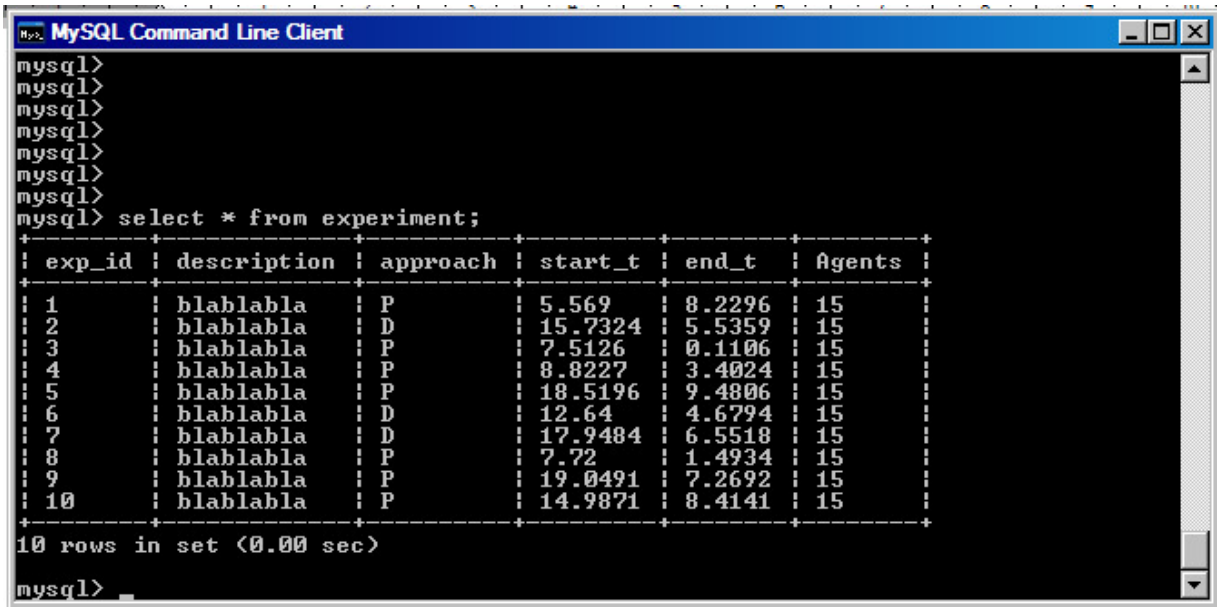

Figure A.4: Experiment Table data

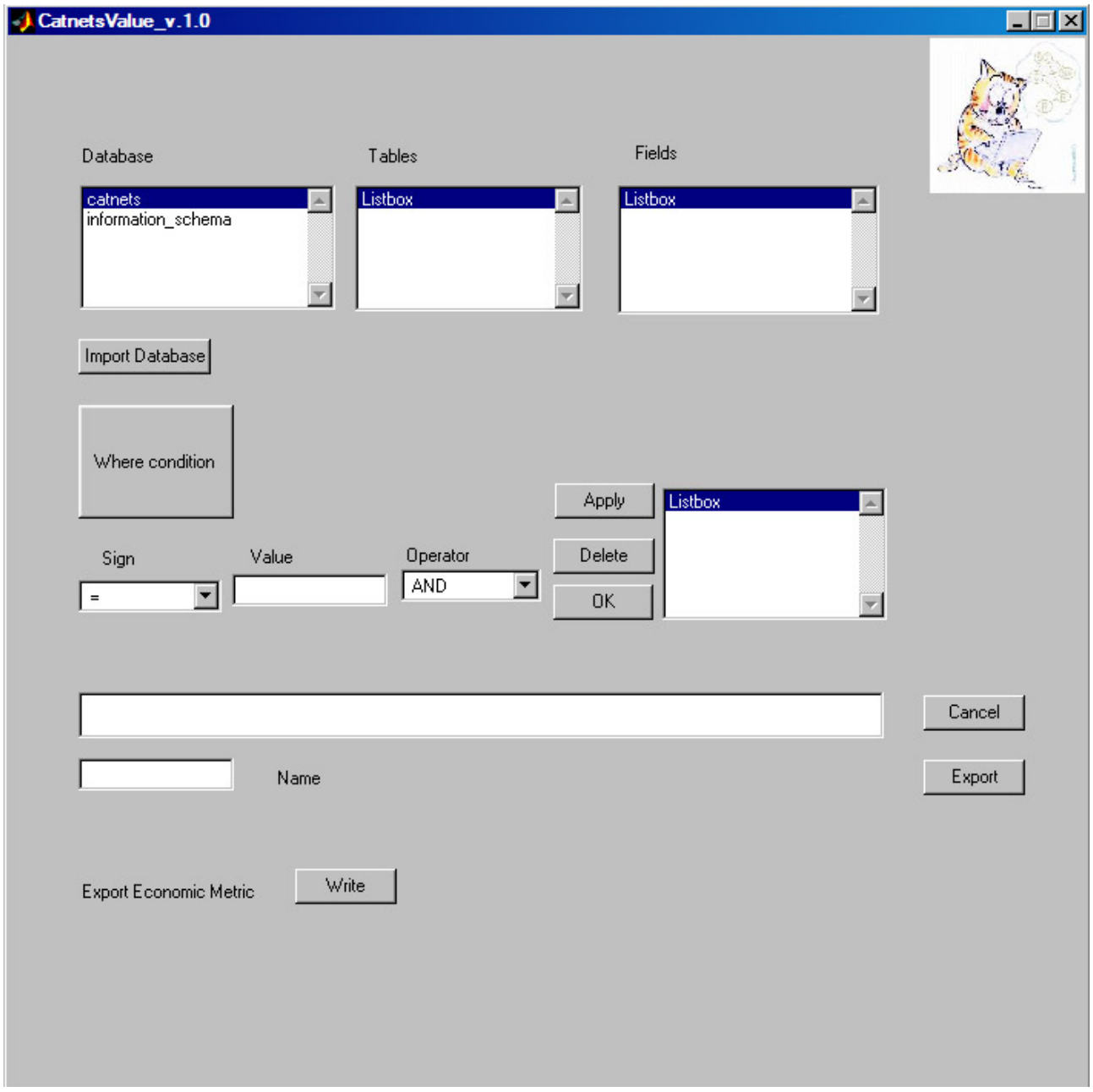

Figure A.5: CatnetsValue GUI

4 Economic metrics

Now we can start the *CatnetsValue* GUI typing into the command window ’catnetsvalue’. The result should be as in Figure A.5. The *CatnetsValue* procedure evaluates the economic metrics, write them into a table called 'metric', and perform a selection of data, on which the evaluation procedures could be carried out.

To evaluate globally the economic metrics, push the 'Write' button. The evaluation of the economic layer metrics is performed for each experiment. Furthermore, an insert operation is performed and each record collects the metrics for each experiment. To see the result type on the Mysql command line client

select * from metrics;

and the table content should be like that in figure A.13 (see last page of this annex).

In order to perform an evaluation process on a subset of data it could be possible to perform an SQL 'select' operation with any criteria and save it as a new table. When CatnetsValue is started the database available for the connection populates the listbox 'Database'. In this case there are the catnets and information schema databases in the listbox. Click twice on the former and click on the 'Import Database' button. The second listbox will be populated by the tables present in the database. In this case the tables are distance, experiment, transactions and usa as in Figure A.6.

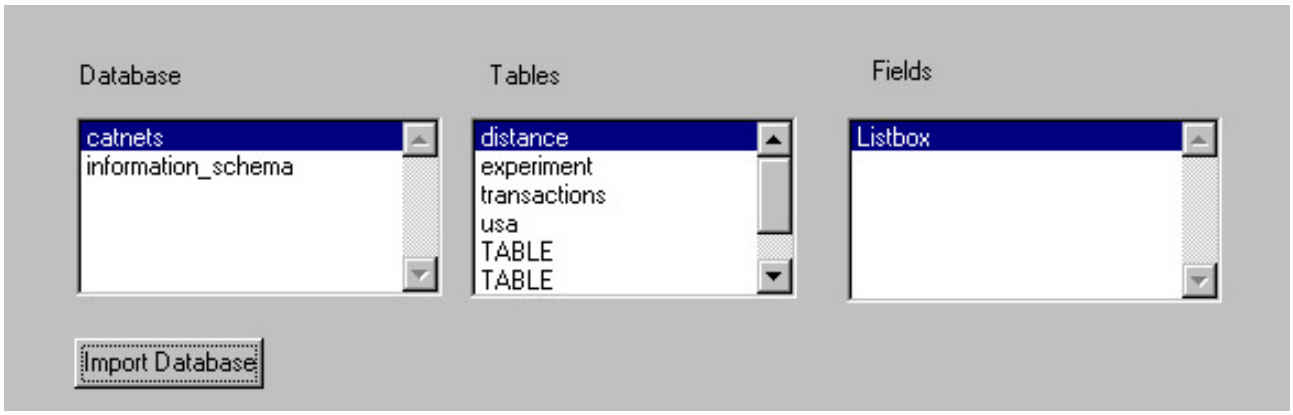

Figure A.6: Import tables function

To observe the structure of one table click on the 'distance' and the column names, which populates the third, listbox as in figure A.7.

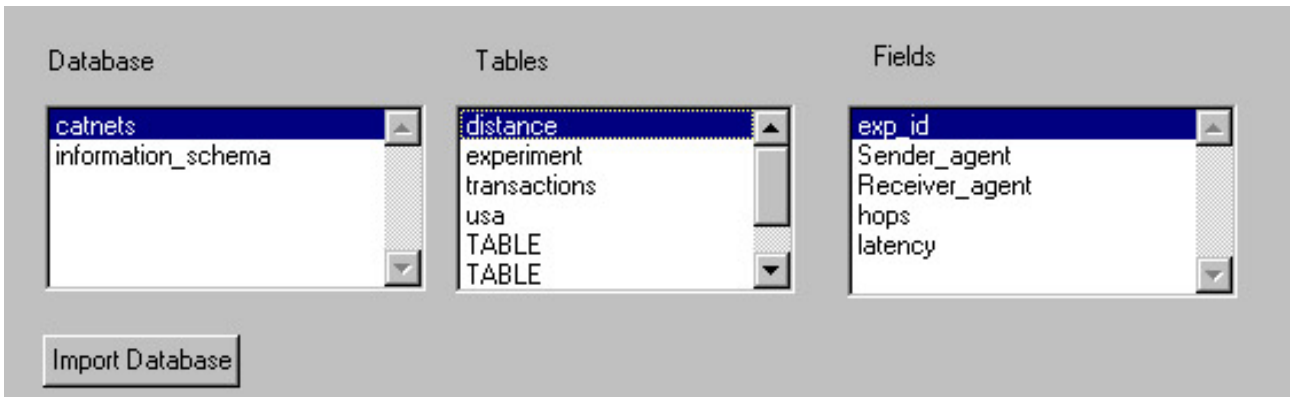

Figure A.7: Import table data function

Click twice on one item in the listbox 'Fields' (Receiver_agent in the figure A.8) and the SQL command *select all Receiver_agent from distance* (figure A.9) is generated.

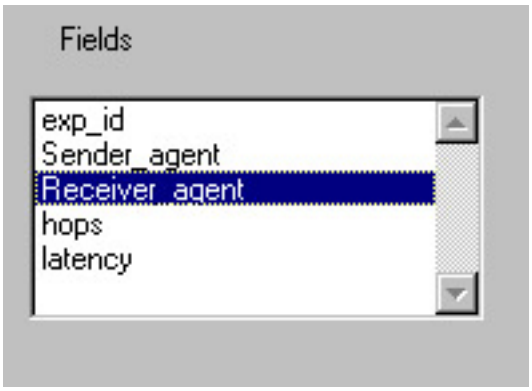

Figure A.8: Receiver_agent field selection

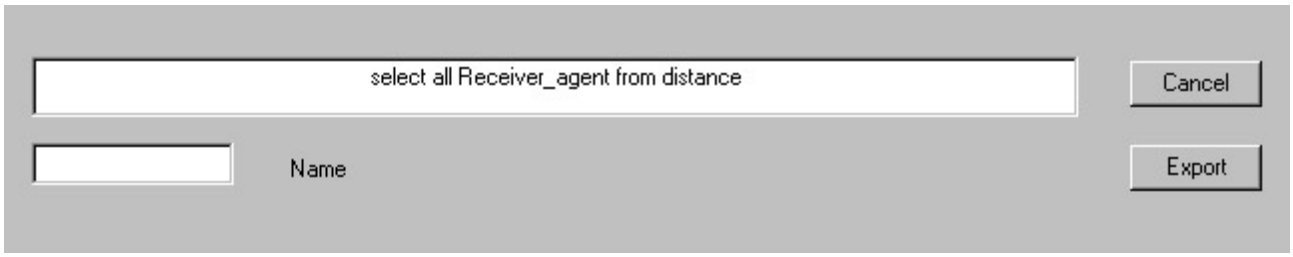

Figure A.9: edit box SQL command

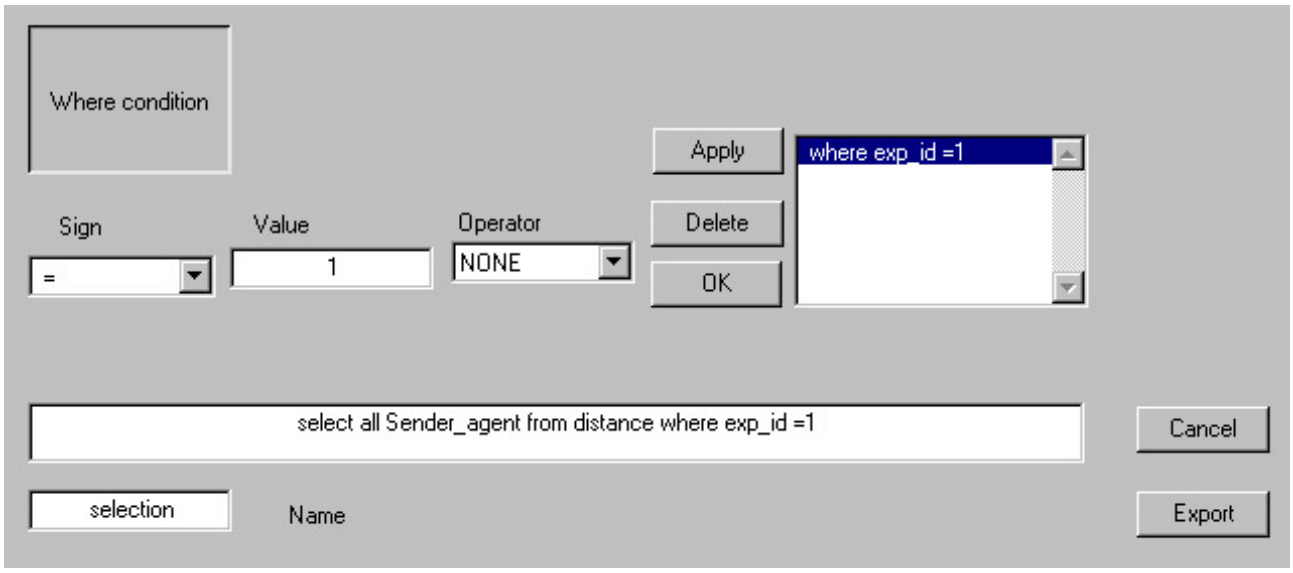

Figure A.10: Where condition setting

5 Multiple Fields selection, where clauses and export

In the section A.4 a simple selection operation is shown. In order to perform more complex operations, *CatnetValue* allows to do a selection on multiple fields and restrict the selection with a 'where criterion'. The simplest example is to select the metrics from each tables with experiment id = 1, i.e select a subset of artificial technical metrics coming from the experiment with id 1. The user has to write into the edit box of the *CatnetValue* the selection command. The SQL command will appear in the edit box following this example. Leave the Toggle button "Where Condition" un-pressed, and select the "Distance" table. Now the third list box is populated by the fields name list of the table "Distance". Select the "Sender_Agent" Field into the third listbox. The result is the print into the edit box of the SQL query "select all Sender_agent from distance". Note that this SQL command does not have WHERE clause. Before introducing it, note that you can manually edit the SQL command or cancel it. Moreover you can include more tables into the selection operation. For example, select two tables, for example Distance and Experiment. Now the result is the presence of the list Distance.exp_id, Distance.Sender_agent, ..., Experiment.exp_id,....., Experiment.end_t in the last listbox (fields). If you select Experiment.exp_id fields, while the result in edit box is:"select all Experiment.exp_id from Distance,Experiment".

Now, if you wish to include the WHERE condition, push the toggle button "Where condition". While it is pushed, it is not possible to alter the SQL command selection, doing new selection on the listbox Tables. Being the WHERE condition button pushed down, select "Experiment" in the listbox Tables and “exp_id” from the "Fields" listbox. Choose the sign "=" from the popupmenu "Sign"; type "1" into the "value" edit box. Select the operator "NONE" from the popupmenu "Operator"; Push the button "apply", into the list box will appear the sentence " where Exp.id = 1". Now push the button " OK". Then you shall have the sentence "select all Sender_agent from distance where exp_id =1".

Finally, you are ready to export the selection into the database. This is done by typing the name of the table into the edit box "name" (type in the name edit box 'selection') click on "export" button and the result is a new table into the Tables database with number and names of columns resulting from the selection query (see figure A.10 for the above instruction). To see the results type in the Mysql command line client:

show tables;

and

select * from selection;

The results are in figures A.11 and A.12.

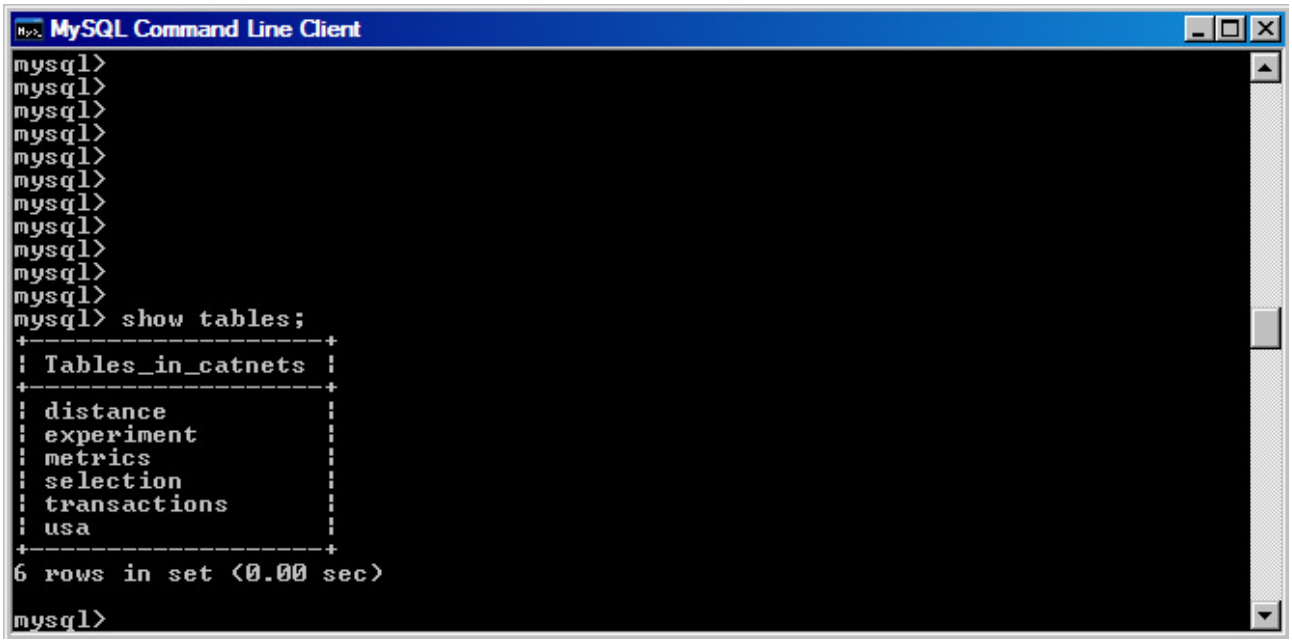

Figure A.11: 'Selection' table

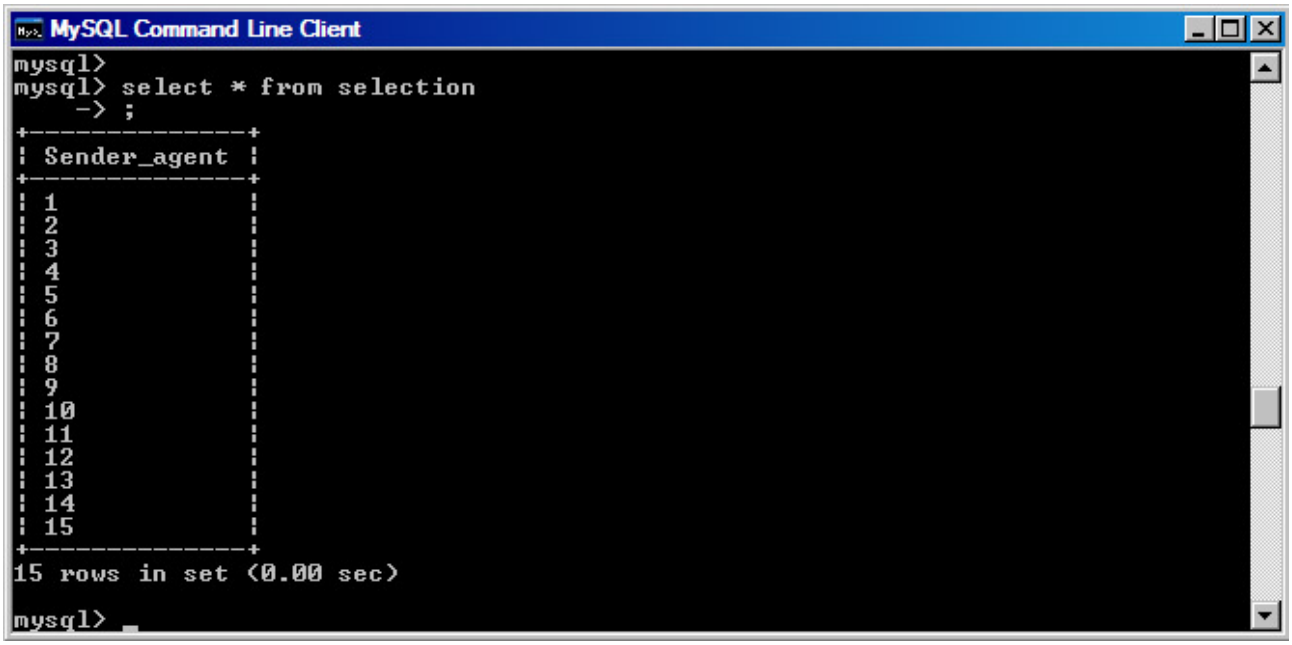

Figure A.12: Selection table content

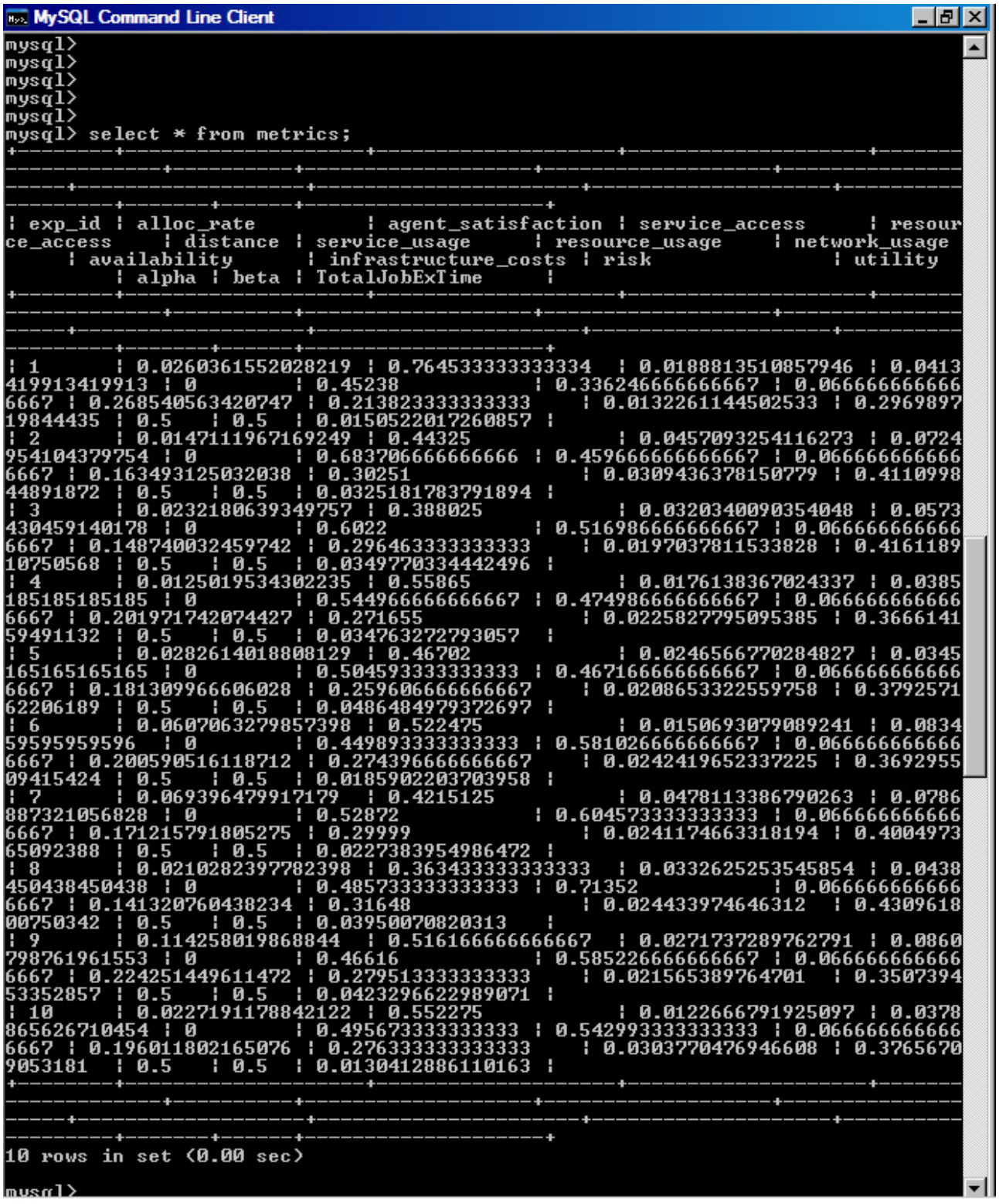

Figure A.13: 'Metrics' table; economic metrics for 10 experiments, 15 agents.