Buss, Georg et al.

Working Paper

Performance Evaluation - Annual Report Year 3

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik, No. 28

Provided in Cooperation with:
University of Bayreuth, Chair of Information Systems Management

Lehrstuhl für
Wirtschaftsinformatik

Information Systems
Management

No. 28

2007

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik

Georg Buss, Nils Parasie, Daniel Veit (University of Mannheim), Michele Catalano (Università delle Marche Ancona), Pablo Chacin, Isaac Chao, Felix Freitag, Leandro Navarro (Universidad Polytecnica de Catalunya), Omer F. Rana, Liviu Joita (Cardiff University), Björn Schizler (University of Karlsruhe), Werner Streitberger, Torsten Eymann (University of Bayreuth)

Performance Evaluation - Annual Report Year 3

Bayreuth Reports on Information Systems Management

UNIVERSITÄT
BAYREUTH

ISSN 1864-9300

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

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

Authors:

Georg Buss, Nils Parasie, Daniel Veit (University of Mannheim), Michele Catalano (Università delle Marche Ancona), Pablo Chacin, Isaac Chao, Felix Freitag, Leandro Navarro (Universidad Polytecnica de Catalunya), Omer F. Rana, Liviu Joita (Cardiff University), Björn Schizler (University of Karlsruhe), Werner Streitberger, Torsten Eymann (University of Bayreuth)

Information Systems and Management Working Paper Series

Edited by:

Prof. Dr. Torsten Eymann

Managing Assistant and Contact:

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

Email: raimund.matros@uni-bayreuth.de

ISSN 1864-9300

IST-FP6-003769 CATNETS

D4.3

Performance Evaluation

Contractual Date of Delivery to the CEC:	31 August 2007
Actual Date of Delivery to the CEC:	04 October 2007
Authors:	Georg Buss, Michele Catalano, Pablo Chacin, Isaac Chao, Torsten Eymann, Felix Freitag, Liviu Joita, Leandro Navarro, Nils Parasie, Omer F. Rana, Björn Schnizler, Werner Streitberger, Daniel Veit
Workpackage:	WP4
Est. person months:	41.5
Security:	public
Nature:	final
Version:	1.0
Total number of pages:	137

Abstract:

This deliverable describes the work done and results obtained in WP4 in the third year of the CATNETS project. Experiments carried out with the different configurations of the prototype are reported and simulation results are evaluated with the CATNETS metrics pyramid. The applicability of the Catallactic approach as market model for service and resource allocation in application layer networks is assessed based on the results and experience gained both from the prototype development and simulations.

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), Chair of Business Administration and Information Systems - E-Business and E-Government / University of Mannheim (Germany).

University of Bayreuth
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: Björn Schnizler
E-mail: schnizler@iism.uni-karlsruhe.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

University of Mannheim
Chair of Business Administration and Informat
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

Changes

Version	*Date*	*Author*	*Changes*
0.1	06/07	FF	Index
0.3	13/07/07	IC	Added prototype-related sections
0.4	26/07/07	FF	Extending sections
0.5	29/08/07	IC/FF	Reorganization
0.5	19/09/07	WS	Evaluation of catallactic allocation approach added
0.6	26/09/07	IC	Pre-Final
0.7	28/09/07	IC	Corrections of section 2 and 4
0.8	29/09/07	WS	Final refinement of all sections
0.9	04/10/07	WS	Section 5.4 and Section 6 added
1.0	04/10/07	WS	Final release

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

This deliverable describes the work done and results achieved in WP 4 of the CATNETS project- In detail, these tasks are finished in year 3 of the work package:

- T 4.1 Metrics specification and implementation, prototype and simulator (Month 7-30)
- T 4.2 Evaluation of implemented market mechanisms (Month 13-30)
- T 4.3 Prototype evaluation (Month 19-30)
- T 4.4 Performance analysis, comparison, evaluation (Month 25-30)
- T 4.5 Further research on properties of Catallaxy applied to computer networks (Month 19-30)

The deliverable reports the performance assessment of the Catallactic approach. It depends on the work done and results achieved in the other workpackages, particularly WP2, the simulator and scenario generator development, and WP3, the prototype development.

Table 1 summarizes the work of WP4 over the three years of the project. The third year targeted on the evaluation of the Catallaxy in two ways: by assessing the developed prototype, and by evaluating the performance of the Catallaxy in several simulation scenarios like comparison to the implemented centralized approach. By the end of the first year, the metrics framework was presented in terms of a metrics pyramid. In the second year, an implementation of a performance measuring framework was achieved.

CATNETS PERFORMANCE EVALUATION	
year 3	Evaluation of the Catallactic mechanism by assessment of the prototype and simulations.
year 1 & year 2	Design of metrics pyramid. Implementation of performance measuring components in prototype and simulator, initial tests of performance measuring infrastructure.

Table 1. Evolution of performance evaluation work in CATNETS

1.1 Structure of the document

The document is divided in four parts: The second chapter recalls the metrics used for assessment of the prototype experiments and the simulations. Compared with previous deliverable, they are now set into the context of how they were used in experimets. Chapter 3 describes the market mechanisms, which were finally implemented in prototype and simulator. Parts of our evaluation are based on the results of the

comparision of the different market mechanism. Chapter 4 describes our assessment of the Catallaxy concerning the feasibility of its implemention in real application layer networks. In chapter 5 our general assessment about Catallxy for resource allocation in application layer networks is presented. Chapter 6 discusses the results obtained. Chapter 7 contains our conclusions.

2 Metrics in prototype and simulator

This chapter describes the metrics finally applied to assess Catallaxy in experiments with the prototype and the simulator.

2.1 Metrics of the prototype

As results from year two of the project, a performance measuring framework was available in order to assess the prototype performance. The measuring framework essentially follows this process:

- During the execution of an experiment, data was periodically or in an event-based way obtained from the three main layers of the prototype (application, middleware, base platform) at each node of the deployed prototype.
- At each node, components of the middleware collected the data from the different layers and wrote it into several local text files.
- After the experiment, the raw data files were collected from the local nodes and moved with the help of scripts (see Annex A) to a global metrics collector.
- Scripts were also used to process the data, compute metrics and to provide a format of the data which could be used for graphical representation.

During the development of the prototype, three kind of economic agents have been implemented. The architecture of the prototype allows to derive from a given base agent class different economic agents. This requires only few code changes for each of agent implemenation. The available agents are:

- Catallactic agents
- Zero Intelligence Plus (ZIP) agents
- Contract Net (CNet) agents

Compared to the Catallactic agents, the ZIP agent and CNet agents are simpler regarding to their configuration and their messaging protocol, easier to use and earlier available. An early version of the ZIP agents includes also real measured resource usage in the price calculations. Later versions of the agents work only with a dedicated resource model. In that model, the price calculation does not take into account the detailed resource usage.

Each agent type generates for the performance measuring components of the middleware several text files. These raw data text files are collected from each node and stored in a central repository. For the three agent types (ZIP agents, CNets agents, and the Catallactic agents), the following data files (Table 2, Table 3, and Table 4) are obtained in the prototype:

FILE NAME	DESCRIPTION
Price.txt	current price of an agent
Match.txt	contains the price at which an offer is accepted ((offer+bid)/2)
Active.txt	logs if the CS is in the market or not (after a successful match an agent leaves the market and returns with the certain probability, for example 1/3)

Table 2. Output files for the ZIP agents

FILE NAME	DESCRIPTION
Price.txt	current price of an agent
sellSucces.txt	contains the number of succesfull trades

Table 3. Output files for the CNet agents

FILE NAME	DESCRIPTION
negotiation_request.txt	CFP received by the BS
negotiation_start.txt	Negotiation start events between 2 agents
negotiation_end.txt	Negotiation end events between 2 agents
strategy_metric.txt	Contains the Catallactic strategy values like current market price (see Table 5 for more information)

Table 4. Output files for the Catallactic agents

In order to allow understanding of the behaviour of the different agents, each agent forwarded data to the middleware which wrote it to particular text files. It was not straightforward to find common parameters which could be obtained in all agent types in the same way. For this reason and also due different time of implementation, there are different text files for each agent type. In order to compare, the raw data of each of agent type has to be processed with scripts a posteriori and off-line, aiming to extract common metrics.

Table 5 illustrates the data written into the *strategy_metric.txt file* of the Catallactic agents. The *strategy metric.txt* file contains the following fields, which provide details on the parameter values used in the agent strategy.

PARAMETER	DESCRIPTION
Agent	Name of the agent
Acquisitiveness	Value for concession level in %
Price Step	Price step size in %
Price Next	Deal range adoption in %
Satisfaction	Satisfaction level of the ongoing negotiation in %
weightMemory	Influence of the price history for the current market price estimation in %
averageProfit	Average profit of the agent
generation	Crossover counter
currentMessageID	ID of negotiation
currentMessagePrice	Agreement price of the negotiation.
currentAverage	Estimated market price
currentLowerLimit	Current lower limit of the deal range
currentUpperLimit	Current upper limit of the deal range

Table 5. Data structure of the agent strategy metrics: the strategy_metric.txt file

From Table 5, it can be observed that the data describing the Catallactic agents contains a large number of parameters. Compared to the other two agent implementrations, there is more information available than for the ZIP and CNet agents. The goal is to find a set of metrics which is available for all three agent implementations.

The *allocationRate* metric was identified as a metric which could be obtained from the data of all the three agent types. It is computed posteriori form the raw data files. The *allocationRate* metric in the ZIP agents is obtained by counting the events in the file *Match.txt* and dividing it by the total number of requests issued till the moment of the metric collection. The *allocationRate* metric in the CNet agents is obtained by counting the events in the file sellSuccess.txt and dividing it by the total number of requests issued till the moment of the metric collection. In order to calculate the *allocation rate* metric in the Catallactic agents, all complex service *negotiation_end* events are counted. Also, the *negotiation_end* events of the BSs are taken into account, since both complex services and basic services can close a negotiation.

The evaluation of the service market only takes the *allocationRate* metric into account because this metric is available in all middleware agent implementations. However, it needs to be noticed, that the Catallactic agent strategy uses a learning mechanism which makes these agents to work in another time scale. On the contrary, the fairly simple decision making in the ZIP and CNet agents could allow obtaining results in experiments with shorter time duration.

2.2 Metrics implemented in the simulator

All metrics - as reported in D4.2 [Del06b] - have been implemented in the simulator. Deliverable D2.3 [Del07a] describes the measured values during a simulation run.

The simulator metrics set is definined following the D4.1 [Del05b]. The complete list of metrics has been refined taking into account two main issues: the simulator development process and the current metric setting available for the prototype. The main changes have been done on the technical level: since the first year's project it was established to collect 12 technical metrics which has been merged together or redefined in the definition. The upper levels (economic, aggregated indicators and final indexes) have been kept unchanged.

Before dicussing the changes, we recapitualte that the technical metrics are used to evaluate two main economic indicators: On DeMand availability (ODM) and Infrastructure Costs (IC). Therefore, technical metrics are divided into two subgroups corresponding to the economic indicators. The ODM group contains the allocation rate, agent satisfaction, discovery time, negotiation time, and service provisioning time. The IC group aggregates number of hops (referred to the distance metric), message size and number of messages, and service and resource usage. The simulator output log provides a larger set of data than the technical metric set requires.

The evaluation process selects the technical metrics defined above and processes them to build a metrics database. The technical metrics at the bottom level of the matrics pyramid are organized per agent and per transaction. The raw data os collected from different simulator output files in the first step. The second step assigns the collected data to the individual agent.
Table 6 shows the available metric set for each agent role. Not all agents have a full set of metrics. But, this scheme holds for each experiment preserving the comparabilty of results.

Metric	CSAgent	BSAgent	RSAgent
Allocation Rate	X	X	X
Satisfaction	X	X	X
Allocation Time	X	X	
Provisioning Time	X		
Distance	X	X	X
Latency	X	X	X
Usage	X	X	
Messages	X	X	X

Table 6. Implemented and evluated metric set for each agent role in the simulator

The following aggregation process is applied for each metric listed in Table 6. We assume a technical metric m_{it} with i as agent index and t as agent transaction. A normalized indicator is computed in a general fashion as

$$I_{it} = f(m_{it}) \qquad \text{and} \qquad f : X \rightarrow Y$$

where $Y \in [0,1]$. The benefit of normalized indicators is twofold: the first benefit is to get interoperability between the different metrics used to compose upper level indicators. This is achieved mainly by normalization to the interval between 0 and 1 which let the metrics leave their initial measurement system units. The second benefit is the ordinal measurement system. We build an ordinal measurement system in which the goodness of system behavior related to the specific metric *m* is better than the value approach. The size of the metric value in absolute numbers is not meaningful any more, and the evaluation and interpretation can only be performed in a realtive fashion, i.e. comparing the same metric for two or more experiments.
The function *f* is specified depending on the individual metric. An exponential function (see Dev05b, pag. 31) is applied for allocation time, provisioning time and usage time:

$$I_{it} = e^{-\beta m_{it}}$$

where β is arbitrarily choosen[1]. The behavior for the metric is depicted in the Figure 1. The problem is to find a function which gives 1 for time value near 0 (this would measure optimality of the behavior system, for example a service provisioning time = 0 it is an ideal and optimal occurrence for the final social utility index) and 0 for large time intervals.

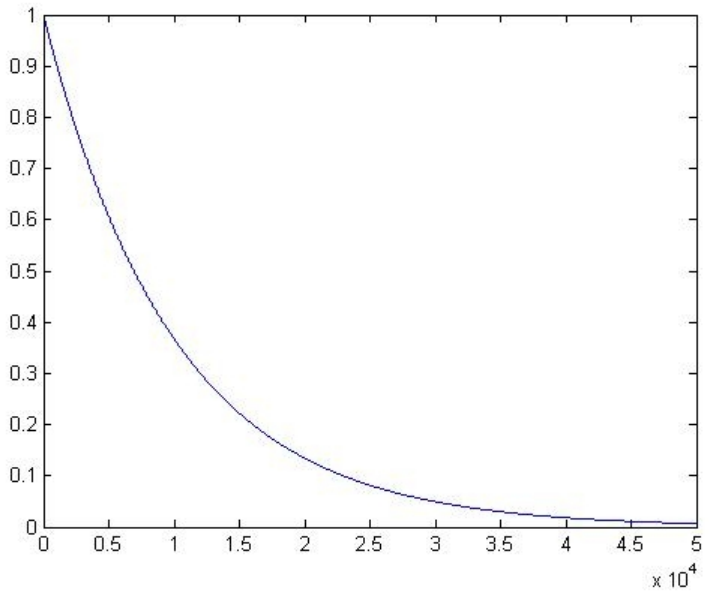

Figure 1– Exponential normalization function between 0 and 50000 milliseconds and the resulting value range; a beta value of 0.0001 is selected for this plot.

The allocation rate is defined as the ratio between the accepted requests and the total number of received request:

[1] The beta parameter defines the curve shape and is fixed for all time metrics and all experiment runs. In particular, the beta value used for the time evaluation is set to $\beta = 0.0001$.

$$Allocation.Rate = \frac{agent.accepts}{agent.requests}$$

The satisfaction is already normalized and calculated during simulation. The calculation is twofold, depending on the seller or buyer role of the agent:

$$satisfaction.buyer_{it} = 1 - \frac{price_{it}}{price.max_i}$$

$$satisfaction.seller_{it} = 1 - \frac{price_{it}}{price.min_i}$$

where price.max and price.min are the price intervals for the agent[2].

The distance metric it is normalized taking into account the number of links between the trading agents. This measure addresses the costs in terms of time and space to trade with longer distance traders. The normalization is performed with repsect to the worst situation for an agent: to trade with an agent at the other side of the network when the topology is a row with all agents:

$$distance_{it} = \frac{links}{\#Agents}$$

Finally the message normalization is done taking the total number of messages[3]:

$$network.usage = \frac{messages_{it}}{\#message}.$$

2.2.2 Economic metric layers

The normalized, technical metrics are taken as input for the economic metric layer. The economic metric layer appregates the metrics using mean and variance of the indicators:

$$E_i = \mu(I_{it})$$

$$\Omega_i = \frac{1}{n}\sum_t \left(m_{it} - E_i\right)^2$$

[2] The price intervals are heterogeneous and can be thougth as demand and supply schedules for the resourcce and service market, as they measure the maximum and minimum price level the agent are willing to trade.

[3] This is the main change in the technical metric level because the metric message size is no more collected. The metric has a constant value for every transaction and agent. This reduces the formula in deliverable D4.2, page 34 to the above one.

where E and Ω are the mean and variance for each agent, respectively. This layer computes the mean values of the metrics for each agent during the simulation run and its variability. The mean and variance indicators are incorporated in the aggregated indexes definded at top of the metric pyramid.
The aggregated economic layer is defined by two indexes: On DeMand availabilty (*ODM*) and Infrastructure Costs (*IC*). Both contain information about the ability of the system to provide the service to a user of the CATNETS allocation approaches and the costs needed to provide them at a high abstraction level.
Renaming the variable as $X = 1 - ODM$ and $Y = IC$, and recalling the fact that X and Y are random variable, the final social utility index is defined as a function of ODM, IC and their variances[4].
In this context, it is needed to evaluate the mean and variance values of agent metrics. They are derived from these formulas:

$$\Pi_j = 1/n \sum_i E_i$$
$$\Phi_j = 1/n \sum_i \Omega_i$$
$$j = 1,...\#metrics.$$

Finally the ODM and IC are obtained by computing the mean and variances:

$$\mu_x = 1 - (1/4) \sum_1^4 \Pi_j \qquad \text{(1-ODM)}$$
$$\mu_y = (1/3) \sum_5^7 \Pi_j \qquad \text{(IC)}$$
$$\sigma_u^2 = (1/4) \sum_{j=1}^4 \Phi_j \qquad \text{(s\_ODM)}$$
$$\sigma_z^2 = (1/3) \sum_{j=5}^7 \Phi_j \qquad \text{(s\_IC)}$$

The final social utility index is

$$L = \alpha\mu_x^2 + \beta\mu_y^2 + \alpha\sigma_u^2 + \beta\sigma_z^2 \qquad \text{(Final).}$$

In brackets behind the formulas, the short names of the values are printed as they are used in the figures of the evaluation section of this deliverable. The alpha and beta weights are set to 0.5 for all evaluations of the allocation approaches. This assumes equal importance of both composite indexes and enables a better comparison of the different scenarios. If

[4] See [Del05b] for details.

one or the other index should be more or less emphasized, a policy maker for a concrete application layer network can adjust the final evaluation function.

2.3 Performance evaluation process in the prototype and the simulator

The performance evaluation process was reported in D4.2 [Del06b]. In year 3, the work continued with adapting and adjusting to practical issues occurring during experiments and observed in the evaluation tasks.

Recalling the main steps, the performance evaluation process of the prototype is made in the following way: The *vxarg* script is used to obtain in parallel the data from the different nodes. Once collected the data, other scripts are used to extract the needed data from the data files. Aggregated values can be computed by Matlab. Graphics are obtained with Matlab and/or GNUPlot. A more detailed description of the particular design of the performance measuring framework on the middleware level of the prototype is described in [FCC2007]. The evaluation process in the context of the simulation environment is presented in deliverable D2.3. [Del07a].

This section presents the main function behavior of the scripts for the analysis of decentralized and centralized behavior in the simulator. The scripts are divided in two packages Catnets_decentral and Catnets_central, which contain the scripts for the scenario evaluation. Both script packages are available on the CATNETS website for doenload. The main behavior scheme could be depicted as in Figure 2:

Figure 2. Main behaviour of scipts

The raw simulator output data is mapped to a structured format, the agent database, by the agent analysis script. Data stored in the agent data base is selected by the agent evaluation script to produce the final index and plot the graphs for data analysis and comparison. In detail, the packages are organized as shown in Table 7.

Catnets_decentral	Catnets_central	Behavior
import_decentral	Import_central	The scripts import the data from txt files and save the metrics in a cell matrix *_mat. Each row contains a dataset of an experiment.
Agentanalysis2	Agentanalysis_c	The scripts select the agents which have traded in the experiment. For each agent, the scripts collect the defined set of metrics and store them in a database like schema called structure.
Agent_eval	Agent_eval_c	These scripts evaluate the upper layer metrics and compute the final social utility index L. Plots are automatically generated which enable the graphically comparison of several simulation runs.
Catnetsplot	Catnetsplot	Using the database structure of the analysis scripts, the catnets plot scripts produce metric plots at agent population level.

Table 7. Scripts package organization

At the end of the evaluation process, there is the final_comparison package available which performs a grahical comparison of the simulations runs between the centralized and decentralized allocation approaches and between several simulation runs of the same scenario.

The main scripts are Agentanalysis2, Agentanalysis_c and agent_eval, agent_eval_c. The agent analysis scripts build the following database structure where the evaluation script can be applied to.

The root element of the structure is test.T1<experiment_id>. Experiment id is the folder, which contains the output files of the simulation run. For example, the root element of the structure for experiment id 184664821646 is test.T1184664821646. The fields of the data structure are organized by the agent id. Agent ids of the simulation follow this format:

```
<agent role> <consecutive agent number for this node> Site
<consecutive node number>
```

An example structure has this format in MATLAB:

```
CSA0Site3: [1x1 struct]
CSA3Site15: [1x1 struct]
BSA0Site9: [1x1 struct]
BSA0Site6: [1x1 struct]
```

```
BSA1Site9: [1x1 struct]
RA1Site29: [1x1 struct]
RA0Site24: [1x1 struct]
RA0Site15: [1x1 struct] …
```

The structure test contains one experiment called T1184664821646. The experiment contains a list of agents. Each agent has its metrics list, which can be accessed using the "." operator in MATLAB.

For example, this operation selects agent CSA1Site8 in the given experiment: test.T1184664821646.CSA1Site8. This results in the metrics list for agent CSA1Site8:

```
Allocation_rate: 0.7579
Satisfaction: [95x1 double]
Allocation_Time: [1x95 double]
Provisioning_Time: [95x1 double]
Distance: [80x1 double]
Latency: [80x1 double]
Usage: [1x80 double]
Messages: 1
```

Further details on the datastructure computation and the source code of the evaluation scripts are moved to the Annex B and Annex C. The whole script packages are available on the CATNETS web site.

3 Evaluation of the implemented market mechanisms

3.1 Market mechanism implemented in the simulator

The market mechanisms implemented in the simulator have been the decentralized Catallactic market mechanism and the centralized approach based on auctions. A detailled description can be found in deliverbale D2.2 [Del06a].

3.1.1 Centralized market

In the simulator, a market for service and resources has been implemented for simulation of the centralized approach. A brief description about practical issues is given here.

Service Market:

For the service market, we implemented a double auction institution [Fri91]. Such auctions are organized by means of order books, each for a set of homogeneous goods. An order book is responsible for storing non-executed orders of the agents. For instance, in the service market there will be n different order books, each for one of the n different services. Buyers and sellers submit their bids in a sealed envelope to the auctioneer. The auctioneer aggregates the bids to form supply and demand curves. Once these curves are aggregated, they are used to set a specific price for trading – the price at which supply equals demand. Double auctions can be either cleared continuously (Continuous Double Auction) or periodically (Periodic Double Auction, Call Market): A Continuous Double Auction (CDA) is a double auction where buyers and sellers simultaneously and asynchronously announce bids and offers. Whenever a new order enters the market, the auctioneer tries to clear the market immediately. A Call Market is a double auction with periodic uniform clearing, e.g. the auctioneer clears the market every five minutes. All orders in a period are collected in an order book and will be cleared periodically. In the implemented component, both clearing can be selected by means of an external parameter.

In the CATNETS simulator, the service market auctioneer is represented as an agent. This auctioneer gets instantiated by the simulator during its initialization and can be contacted by every other agent. Complex service agents and basic service agents communicate with the auctioneer by means of messages, i.e. they can submit their bids in form of messages. Furthermore, they can receive further information from the auctioneer agent such as the current market price. In case the auctioneer cleared the market – i.e., it computed an outcome and prices – agents get informed whether or not they are part of the

allocation. A detailed description of the integration can be found in Deliverable 2.2 [Del06a].

Resource Market:

In the resource market, participants are the basic services as resource consumers (buyers) and resource services (sellers) offering computational services having specific capacities, e.g. processing power. The same resources (e.g. CPUs) can differ in their quality attributes, e.g. a hard disk can have 30GB or 200GB of space. An adequate market mechanism for the resource market has to support simultaneous trading of multiple buyers and sellers, as well as an immediate resource allocation. Furthermore, the mechanism has to support bundle orders – i.e. all-or-nothing orders on multiple resources – as basic services usually demand a combination of computer resources. For comprising the different capacities of the resources (i.e. resources can differ in their quality), the mechanism has to support bids on multi-attribute resources.

Reviewing the requirements and surveying the literature, no classical auction mechanism is directly applicable to the resource market. Instead, a multi-attribute combinatorial exchange (MACE) is applied that satisfies the described requirements [Sch07].

MACE allows multiple buyers and sellers simultaneously the submission of bids on heterogonous services expressing substitutabilities (realized by XOR bids) and complementarities (realized by bundle bids). Furthermore, the mechanism is capable of handling cardinal attributes as well as an immediate execution of given orders as the clearing can be done continuously. For instance, a resource consumer can bid on a bundle consisting of a computation service and a storage service. The computing service should have two processors where each processor should have at least 700MHz. Furthermore, the storage service should have 200MB of free space. After the participants submitted their bids to the auctioneer, the allocation (winner determination) and the corresponding prices are determined.

The resource market is integrated similarly into the CATNETS simulator as the service market. The auctioneer is represented as an agent and has access to the market implementation. A detailed description of the integration can be found in Deliverable 2.2 [Del06a].

3.1.2 Decentralized market

This section describes an alternative, decentralized approach. The bargaining mechanism introduced here, implements the selection decision in the requesting client itself. Related realizations of decentral approaches are found in P2P Networks, where Gnutella [AH00] is a typical example. An optimization of network performance is out of the scope of the clients behavior; in contrast, the selfish conduct of each peer leads to performance and congestion problems in the P2P network, which are principally hard to solve [AH00].

Gnutella uses a flooding algorithm for service discovery. The catallactic approach also uses flooding for decentral service and resource discovery.

In decentral matchmaking models, agents communicate directly with each other, decide on their own, and do not take the system state into account. In the Edgeworth process [Var94] economic subjects trade bilaterally with each other only if their utility is supposed to increase after the barter. In that case, the sum of all utilities increases after each successful barter; the final state is Pareto-optimal and has maximum system utility. A theoretical fundament for how dynamic market processes, heterogeneous agents and choice under incomplete information work together can be found in Neo-Austrian Economics, in particular in Friedrich August von Hayek's Catallaxy concept [HBK+89]. Catallaxy describes a state of spontaneous order, which comes into existence by the community members communicating (bartering) with each other and thus achieving a community goal that no single user has planned for. The implementation of Catallaxy uses efforts from both agent technology and economics, notably agent-based computational economics [Tes97].

An iterative bilateral negotiation protocol, similar to a contract-net, is used since no complete information is available [ST98]. Both agents approximate to the trade-off point in iterative steps exchanging offers and counter-offers. This process is described as monotonic concession protocol [RZ94]. If an agent receives an offer or counter-offer, it decides to either make a concession or send the same price as in the last negotiation until the negotiation ends with an accept or a reject. After the negotiation, the autonomous agents adapt their negotiation strategies using a feedback learning algorithm. The learning concept used in this simulation is derived from so-called gossip learning. This means that the agents learn from received information about other transactions in the market. This information may not be accurate or complete, but serves as an indication about the gross direction of the market. In our implementation, this gossip information is created and broadcast by a successful agent, in analogy to issuing an adhoc information in stock market periodicals. In economic simulations lots of research efforts on evolutionary algorithms can be found. We selected the STDEA (Smith Taylor Decentral Evolutionary Algorithm) [ST98]. The STDEA is a decentral evolutionary algorithm, which has no global evaluation metric (fitness value), used in genetic algorithms to separate the under performing participants [Gol93]. A fundamental quality of the mechanism is the decentral communication and fitness evaluation, using local available data. Every agent sends a plumage object after a successful transaction, advertising its average income (fitness) and its genes (genotype) to all agents of the population after an evaluation phase, i.e. after it has carried out a certain number of negotiations with this genotype. If an agent receives a plumage object from other agents, it decides using a blindness probability, whether the plumage objects is evaluated, avoiding premature unification of the genotype.

Sender and recipient remain anonymous. If a certain maturity threshold of received plumage is exceeded, the agent replaces his old genotype with the evolved version after the completion of evaluation, selection, recombination and mutation phases as in normal

genetic algorithms. The mutation rate is also influencing the algorithm, which determines the frequency and the extent of explorative behavior of the population.

Ongoing communication by using price signalling leads to constant adaptation of the system as a whole and propagates changes in the scarcity of resources throughout the system. The resulting patterns are comparable to those witnessed in human market negotiation experiments [KR95] [ST99] [Pru81].

3.2 Evaluation of the market mechanism implemented in the simulator

3.2.1 Comparision of the centralized and the decentralized allocation approach

The objective of this section is the performance comparison of the centralized and the decentralized allocation approach. Therefore, two sets of scenarios where developed. The goal of the first set is to evaluate how the centralized and the decentralized approach deal with a raising number of agents within a fixed large network topology. The second set of scenarios is designed to evaluate how the density of agents within a network topology affects the outcomes of both mechanisms.
In Section 3.3, the different types of services, their relation to each other and the market property files are introduced. This service configuration remains the same all simulation runs in both scenario sets. In Section 3.4, the scenarios from the first set and the second set are described in detail. The description of the different experiments and their evaluation is presented in Section 3.5 and Section 3.6.

3.3 Services, Resource Types and Market Configuration Files

The service types on service and resource markets are the same for both sets of scenarios. Three complex service types, four basic service types and three resource service types are specified. In detail, these are CS1, CS2, CS3, BS1, BS2, BS3 as well as ARB1, ARB2, ARB3. The dependences between the services are depicted in Figure 3.

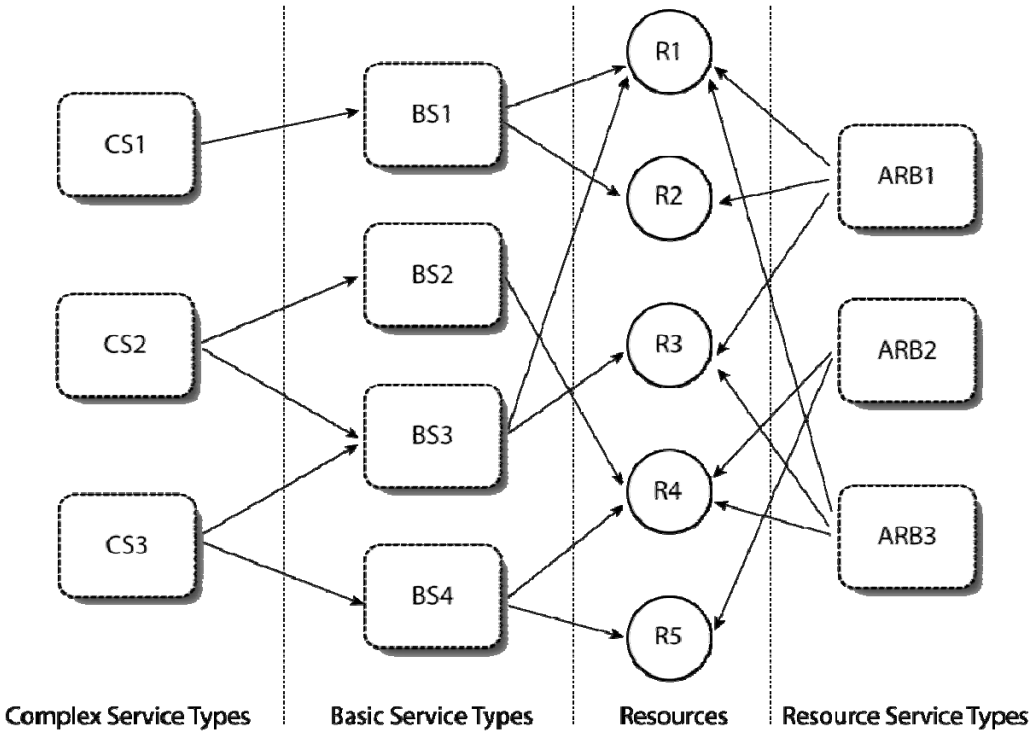

Figure 3. Service types and their dependencies

For example, a complex service of the type CS1 requires a basic service of the type BS1 in order to perform its task. Each type of basic service needs some resources to perform its tasks. In the specific case of BS1, these are the resources r1 and r2. These resources are at most partially provided by the resource service types ARB1 and ARB2. For the complete set of configuration files (arb.conf, bs.conf and cs.conf) the reader is referred to the example package of the simulator release. This is available for download on the CATNETS website.

The same market configuration files are used in all scenarios. These files are market_decentral.properties, strategy.conf, and learning.conf for the decentralized case and market_central.properties for the centralized case. The reader is referred to deliverable D2.3 for a detailed parameter description of those files. In the market_decentral.properties file, the starting price ranges for buyers and sellers on the service as well as the resource market are specified as depicted in Table 8.

Basic service price ranges	Resource product price ranges
bs1.seller.minPrice = 55 bs1.seller.maxPrice = 65 bs1.buyer.minPrice = 55 bs1.buyer.maxPrice = 65 bs1.hard.lower.limit = 25 bs1.hard.upper.limit = 85 bs1.resource.itemids = r1r2_0	r1r2_0.seller.minPrice =50.0 r1r2_0.seller.maxPrice =60.0 r1r2_0.buyer.minPrice =50.0 r1r2_0.buyer.maxPrice =60.0 r1r2_0.hard.lower.limit =20.0 r1r2_0.hard.upper.limit =80.0 r1r2_0.baseunit.r1= 1 r1r2_0.baseunit.r2= 1 r1r2_0.resourceids = r1 r2
bs2.seller.minPrice = 30 bs2.seller.maxPrice = 35 bs2.buyer.minPrice = 30 bs2.buyer.maxPrice = 35 bs2.hard.lower.limit = 15 bs2.hard.upper.limit = 45 bs2.resource.itemids = r4_0	r1r3_0.seller.minPrice =50.0 r1r3_0.seller.maxPrice =60.0 r1r3_0.buyer.minPrice =50.0 r1r3_0.buyer.maxPrice =60.0 r1r3_0.hard.lower.limit =20.0 r1r3_0.hard.upper.limit =80.0 r1r3_0.baseunit.r1= 1 r1r3_0.baseunit.r3= 1 r1r3_0.resourceids = r1 r3
bs3.seller.minPrice = 55 bs3.seller.maxPrice = 65 bs3.buyer.minPrice = 55 bs3.buyer.maxPrice = 65 bs3.hard.lower.limit = 25 bs3.hard.upper.limit = 85 bs3.resource.itemids = r1r3_0	r4_0.seller.minPrice =25.0 r4_0.seller.maxPrice =30.0 r4_0.buyer.minPrice =25.0 r4_0.buyer.maxPrice =30.0 r4_0.hard.lower.limit =10.0 r4_0.hard.upper.limit =40.0 r4_0.baseunit.r4= 1 r4_0.resourceids = r4
bs4.seller.minPrice = 55 bs4.seller.maxPrice = 65 bs4.buyer.minPrice = 55 bs4.buyer.maxPrice = 65 bs4.hard.lower.limit = 25 bs4.hard.upper.limit = 85 bs4.resource.itemids = r4r5_0	r4r5_0.seller.minPrice =50.0 r4r5_0.seller.maxPrice =60.0 r4r5_0.buyer.minPrice =50.0 r4r5_0.buyer.maxPrice =60.0 r4r5_0.hard.lower.limit =20.0 r4r5_0.hard.upper.limit =80.0 r4r5_0.baseunit.r4= 1 r4r5_0.baseunit.r5= 1 r4r5_0.resourceids = r4 r5

Table 8: Initial price configuration for the services and products traded on the service and resource market

The left column of the table contains the valuations for the basic service types the service market participants start with. The right column lists the valuations for the products on the resource market resource market participants start with. The starting price ranges are the same for basic service buyers and sellers. The valuations depend on the product assigned to the basic service types. If a product consists of two resource types, the price ranges almost double (cf. bs4 and bs2). The same configuration model is applied to the resource market. This guarantees that the product r4_0 consisting of the resource r4 cannot be more valuable than the product r4r5_0 consisting of the resources r4 and r5.

The hard lower and the hard upper price limits are set on the service market 5 units above the corresponding limits on the resource market. This models value creation between the two markets. A basic service type should have at minimum the same value than the sum of resource types it is consuming have. In the learning.conf file, the strategy parameters used for the comparisons were defined as depicted in Table 9.

Strategy
maturityThreshold = 5
courterThreshold = 20
mutationProbability = 0.05
ringSize = 10000
crossOverSelectionModel = 0
gaussWidth = 0.01 min = 0.001 max = 0.999
genotype.randomize = no
genotype.acquisitiveness = 0.05 genotype.satisfaction = 0.99 genotype.priceStep = 0.5 genotype.priceNext = 0.05 genotype.weightMemory = 0.9

Table 9: Strategy parameter set

Agents using that strategy are likely to continue negotiations making high concessions. That leads to fast negotiation rounds that are not likely to be aborted.

In the market_central.properties file, the parameters were set as depicted in Table 10. The imitate strategy parameter was set to zero. A simpler version of the valuation generator was used. The valuations of the agents were drawn form a normal distribution with a mean of 10 and a deviation of 1. The lower limit for the values to be generated was set to 2.85 units. It was verified by the evaluation of simulation runs, that this configuration models best the strategy applied by the agents in decentralized case. LPSolve was used as solver and the search for disjunctive sets within the orderbooks was switched off. The time limit of the solver was set to 1200ms. For a detailed insight into the configuration files themselves the reader is referred to the simulator package.

Central market configuration
basic.useServiceMarketPrice = 1
service.kprice = 0.5
resource.kprice = 0.5
resource.numberattributes = 1
resource.updateunsuccessful = 0
resource.orderbook.finddisjunctivesets = false

resource.orderbook.split = 0
resource.allocator.model = 3
resource.allocator.solver = 1
resource.allocator.timelimit = 1200
valuation.imitateStrategy = 0
valuation.smallestvalue = 2.85
valuation.normal.mean = 10
valuation.normal.deviation = 1

Table 10: Parameter valus of the central market configuration file

3.4 Scenarios

The scenarios of the first set are developed to evaluate how the centralized approach and the decentralized approach deal with a rising number of agents within a large topology. Three scenarios are created. Each of those scenarios is based on the same network topology with 500 nodes, which are partially connected. If there is a link between two nodes, the transmission rate has a minimum capacity of 1024 Mb/s. The probability of node failure is set to zero. The agents are randomly distributed in each scenario. 20% of the total agent number are complex service agents, 40% are basic service agents and 40% are resource service agents. A complex service agent is able to handle each type of complex service request. The basic and resource service agents are dedicated to a specific service type. The number of these types was uniformly distributed. The scenarios were defined as follows:

1. 100A_500N: 100 agents within a topology of 500 nodes
2. 200A_500N: 200 agents within a topology of 500 nodes
3. 300A_500N: 300 agents within a topology of 500 nodes

For a detailed view on the network topology and the precise location of each service within the topology, the reader is referred to the example package on the CATNETS website.

The scenarios of the second set are developed to evaluate how the centralized approach and the decentralized approach deal with the same number of agents within topologies differing in size. Again, three scenarios are created whose topologies have up to 50 nodes. This network is also partially connected; not all nodes are connected to each other like in a fully connected mesh. The links have a constant maximum bandwidth of 1024 Mb/s. The nodes' failure probability is zero. The agents are randomly distributed on the nodes in each scenario. 20% of the total agents' number is complex service agents, 40% are basic service agents and 40% are resource service agents. A complex service agent is able to handle each type of complex service request. The basic service and resource service agents are dedicated to a specific service type. The number of agent types is uniformly distributed. The scenarios are defined as follows:

1. 50A_10N: 50 agents within a topology of 10 nodes
2. 50A_30N: 50 agents within a topology of 30 nodes
3. 50A_50N: 50 agents within a topology of 50 nodes

For a detailed view on each network topology and the precise location of each service within the specific topology, the reader is referred to the example package on the CATNETS website.

3.5 Experiments Scenarios 1

The goal of the experiments is to evaluate how the centralized and the decentralized mechanism perform in scenarios with a topology of 500 nodes with up to 300 agents.
During the third year, we are not able to achieve meaningful simulation results for the scenarios of the first set in centralized case. The reason is the not properly working advanced Grid time model of the simulator for the centralized allocation approach. Instead of using the advanged Grid time model, only the real time model could be used for simulations. This extends the duration of simulation runs a lot in comparison to decentralized case.
It was planned to perform simulation runs with at least 10000 requests. But, even a single simulation run with only 1000 requests lasts up to one week depending from the number of agents placed in the topology. It is not possible to calibrate the simulator parameters in a manner that a meaningful comparison could have been achieved. Therefore, there is no analysis part in this section.

3.6 Experiments Scenarios 2

The experiments presented in this section are all based on the second scenario set. In Section 3.6.1, the simulator configuration, which is used for each experiment, is described. The results of the experiments are used to gain experience of how the centralized as well as the decentralized mechanism perform in different scenarios (3.6.2). In Section 3.6.3, the results of centralized and decentralized simulation runs are compared to each other. In Section 3.6.4, the influence of the hopcount parameters on the decentralized allocation approach is evaluated. Simulation runs with different hopcount values are analyzed.

3.6.1 Simulator Configuration

Each experiment is started with 1000 complex service requests. The complex service selection probability is the same for all complex service types. Demand is submitted randomly to the complex service agents. The time interval between the submissions of complex service requests is set to 1000 milliseconds. The queue size, which indicates how many requests a complex service agent is able to store, is set to 2000. This ensured that no request is lost. The basic service execution time is set to 100 milliseconds. Both markets are connected. The budget of a basic service buyer is limited by the earnings it

has achieved on the service market. The negotiation timeout is set to 5000 milliseconds and the message size to 2 kByte. A small message size avoids transmission problems which are out of scope in this scenario.

For the centralized market mechanism, the clearing policy of the service market is set to a continuous double auction. The continuous double auction is chosen because the simulator is not able to perform simulations in call market mode if the advanced Grid time model is switched on. On the resource market, the clearing policy is set to a call market. The according clearing interval is set to 400 milliseconds.

The parameters for the decentralized market mechanism are set as follows: The starting price ranges are not randomized. The dedicated resource model is used and the proposals are selected according to the option: best price – one shot. Co-allocation is switched off. The parameter hop count is set to one hop count.

3.6.2 Comparison of centralized and decentralized simulation results

In this section, centralized and decentralized simulation runs are analyzed separately from each other. The goal is to evaluate whether the results differ for topologies of varying size. 10 simulation runs are performed in the centralized as well as in the decentralized mode for each scenario. The overall results of the simulation runs performed in the centralized mode are depicted in Figure 4, Figure 5, and Figure 6.

Figure 4: Final (social utility index) bar diagram centralized comparison of 50 agent and different topologies; 10 simulation runs for each scenario are plotted.

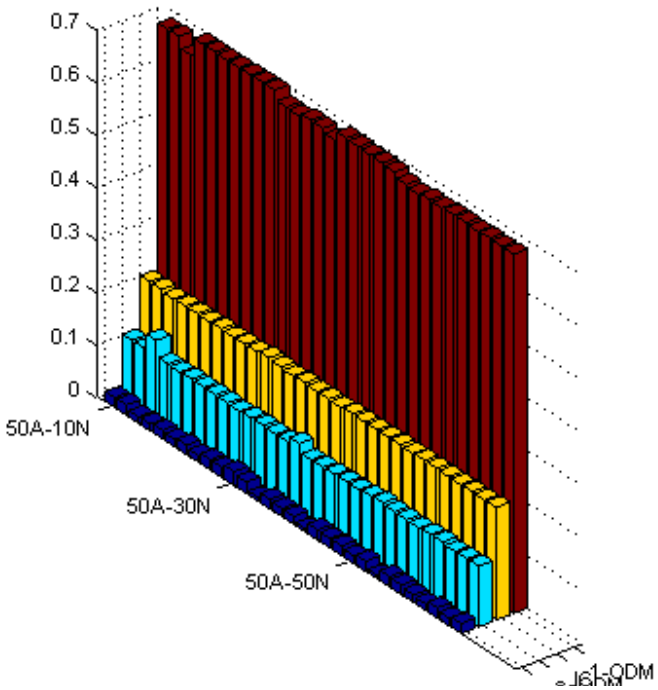

Figure 5: ODM and IC for centralized comparison

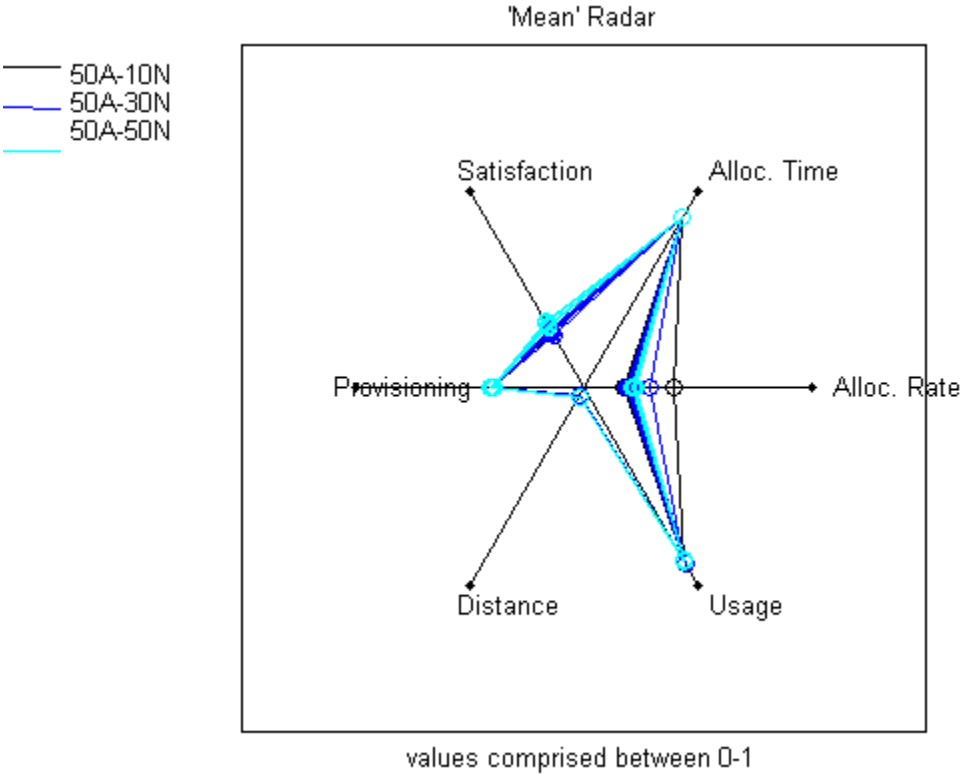

Figure 6: Mean spider centralized comparison

For the centralized mode (three scenarios, ten runs each), the values are almost stable even if the topology size changes. The final index is computed by the inverse On DeMand availability (1-ODM), the Infrastructure Costs (IC) as well as the related standard deviations. These values are depicted in Figure 5. They do also not change significantly. A high inverse On DeMand availability (1-ODM) and low infrastructure costs can be observed.

Figure 6 shows the mean and the standard deviation of the values the IC and the 1-ODM are computed of. The runs performed for a specific scenario are plotted in the same color. The figures show that the mean values as well as the related standard deviations differ slightly for the different scenarios. Only two simulation runs differ significantly in the allocation rate.

The small deviations of the overall results imply that the density of agents within a topology does not influence the performance of the centralized mechanism. This is an obvious observation for a market mechanism where supply and demand are coordinated by a central auctioneer. The low IC value can be explained by the short distances between the auctioneer and each node that does not deviate (Figure 7). The low value of the distance parameter flattens the second influencing value of IC, which is usage. The high usage value shows that an agent has almost no idle times – it is biding or delivering a service. The high value of the allocation time parameter indicates that agents spend low time on allocation of service and resources. The solver computes very fast the allocation. This parameter drives the (1-ODM) value.

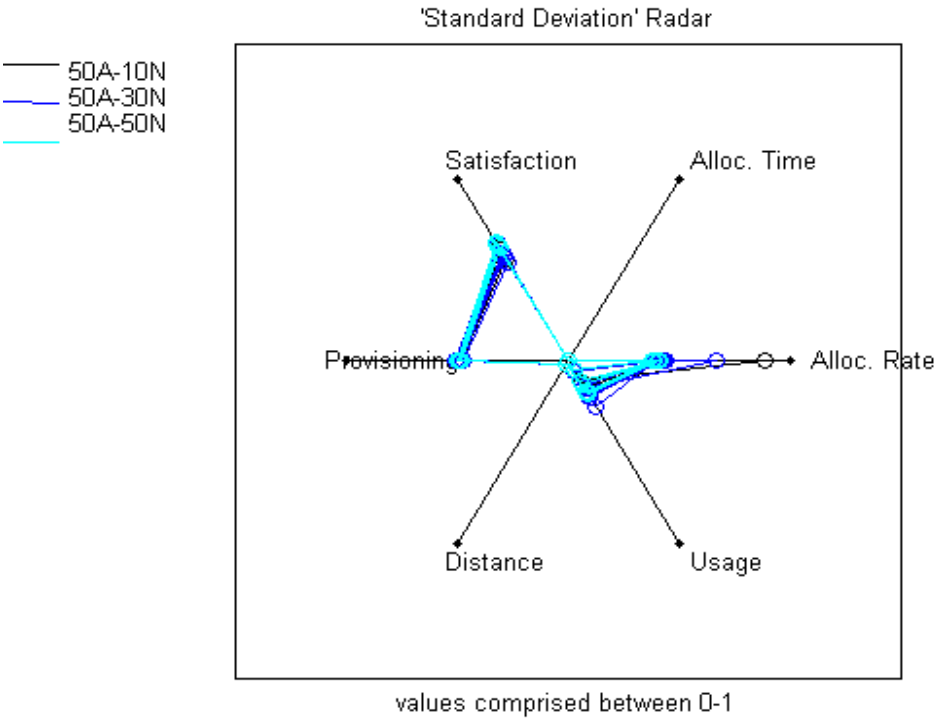

Figure 7: Standard deviation spider centralized comparison

The overall results of the decentralized mode are depicted in Figure 8,Figure 9,Figure 10 and Figure 11.

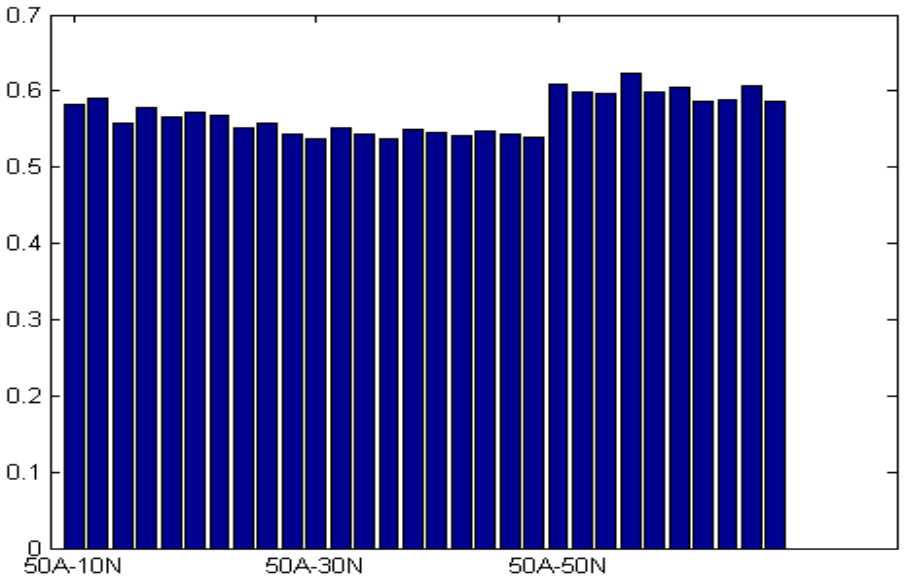

Figure 8: Final bar decentralized comparison

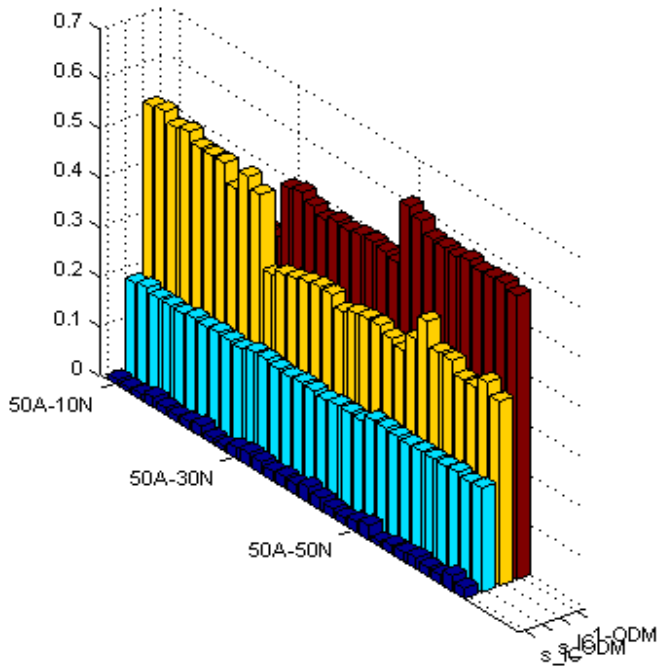

Figure 9: ODM and IC decentralized comparison

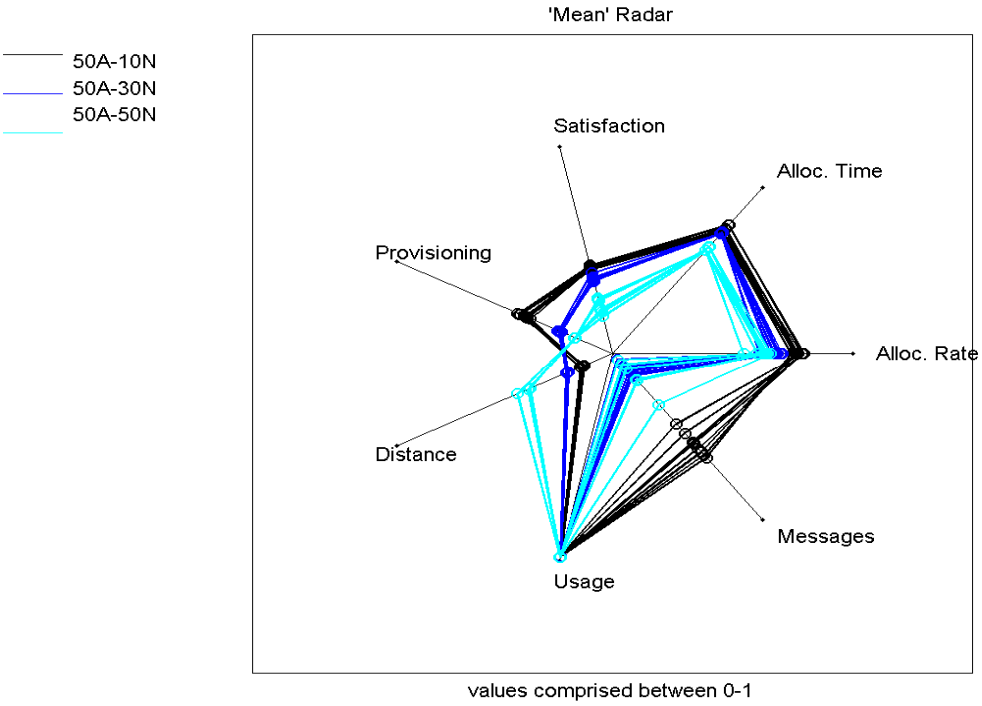

Figure 10: Mean spider decentralized comparison

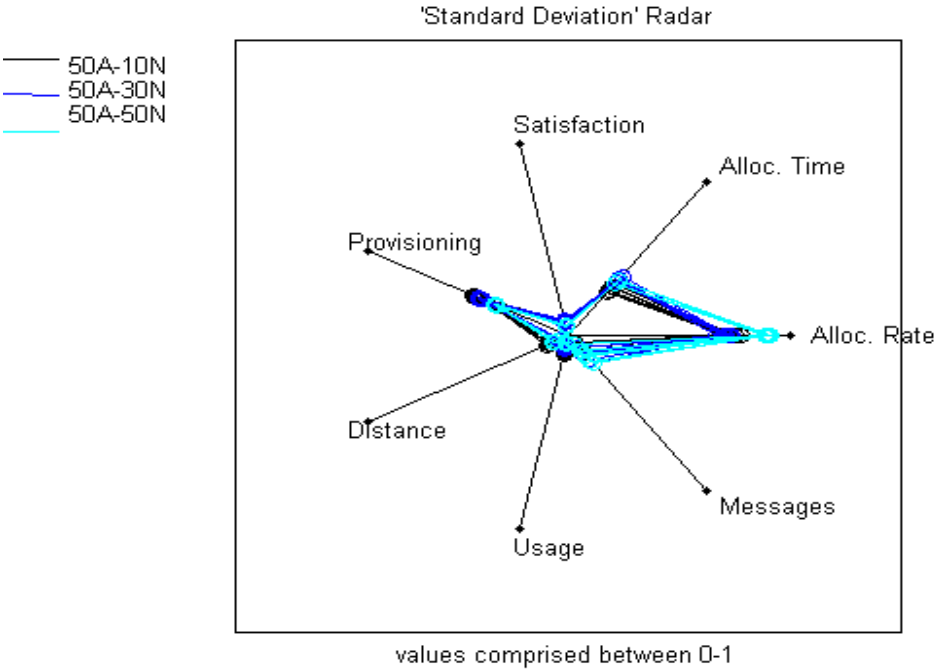

Figure 11: Standard deviation spider decentralized comparison

Figure 8 shows the final index computed for each single simulation run executed in the decentralized mode (three scenarios, ten runs each). The final values are slightly fluctuating between the runs of the same scenario. Furthermore the final values change significantly if the topology size changes. The final index is computed by the mean invese on demand availability (1-ODM), the mean infrastructure costs (IC) as well as the related standard deviations. These values are depicted in Figure 9. They also change significantly between different scenarios. In the scenario 50A_10N, a lower inverse on demand availability and high infrastructure costs can be observed, whereas this changes for the scenarios 50A_30N and 50A_50N. Here, the graph shows a high inverse on demand availability (1-ODM) and low infrastructure costs.

The related standard deviation values are constant over all simulation runs. Figure 10 and Figure 11 show the mean and standard deviation values the IC and 1-ODM values are computed of. The runs which correspond to a specific scenario are plotted in the same color. The figures show a significant change in the mean values for the runs of different scenarios. The related standard deviation values differ only slightly. Only the values of the allocation rate deviate a bit.
The deviations of the final values between the different scenarios show that the topology size influences the outcome of the decentralized approach. Two effects can be observed if the topology size is increased. On the one hand the number of negotiation partners decreases. This is caused by the parameter hopcount which limits the range of the call for proposal message. The decreasing number of negotiation partners results in decreasing IC (Figure 9). On the other hand in a bigger network topology the 1-ODM increases. This is mainly caused by the decreasing number of negotiation partners. The less the negotiation partners are available the higher the probability is that they are busy. This is verified by the values IC and 1-ODM are computed of. The bigger the topology the more the numbers of sent messages declines. Agein, the reason is that the number of possible negotiation partners decreases, whereas the distance value rises a little bit and the usage parameter is constant. That causes a declining IC value. The 1-ODM value decrease is caused by a lower allocation rate as well as a lower satisfaction. The best final result using the decentralized mechanism was achieved in the scenario 50A_30N. In that case, IC and 1-ODM are balanced best.

3.6.3 Comparison of centralized to decentralized simulation results

In this section, the centralized and the decentralized mechanism are compared to each other by means of the results of 10 simulation runs for each scenario. The goal is to evaluate which mechanism performs better in which scenarios.
Like Figure 4 and Figure 8 from section 3.6.2, they show that the centralized mechanism performs better in each scenario. The final social utility index value is almost constant around 0.51, whereas the final value for the decentralized mechanism fluctuates between 0.61 and 0.53. The better performance of the centralized approach is based on the lower IC costs compared to the decentralized approach. The On DeMand availability of the resources is higher for all runs performed in decentralized case (Figure 5 and Figure 9).

But, the mean IC and their standard deviation are higher in decentralized simulation runs. The overall results show the centralized mechanism outperformes the decentrlaized approach.

Figure 12, Figure 13, Figure 14, and Figure 15 show a comparison of the results of the simulation runs for the scenario 50A_30N. This scenario is chosen for the direct comparison of the centralized to the decentralized approach because it showed the best results for decentralized case. Figure 12 depicts the final values for each simulation run. It can be observed that the decentralized case results are slightly worse than the ones of centralized case. This is caused by the high mean IC of decentralized case compared to the ones of centralized case (Figure 9). Furthermore the better 1-ODM mean value of decentralized case is not able to turn the tide because of its high standard deviation. That is what Figure 14 and Figure 15 depict in detail. The mean allocation rate, which drives odm, is really high for the decentralized case compared to the centralized approach (Figure 14). But it is combined with a high standard deviation (Figure 15). That is why the decentralized case cannot outperform the decentralized case. The high mean allocation rate which drives the ODM value is neutralized by its high standard deviation. Moreover the IC values usage and distance turn out better for the centralized case. That combination makes the centralized case superior to the decentralized case.

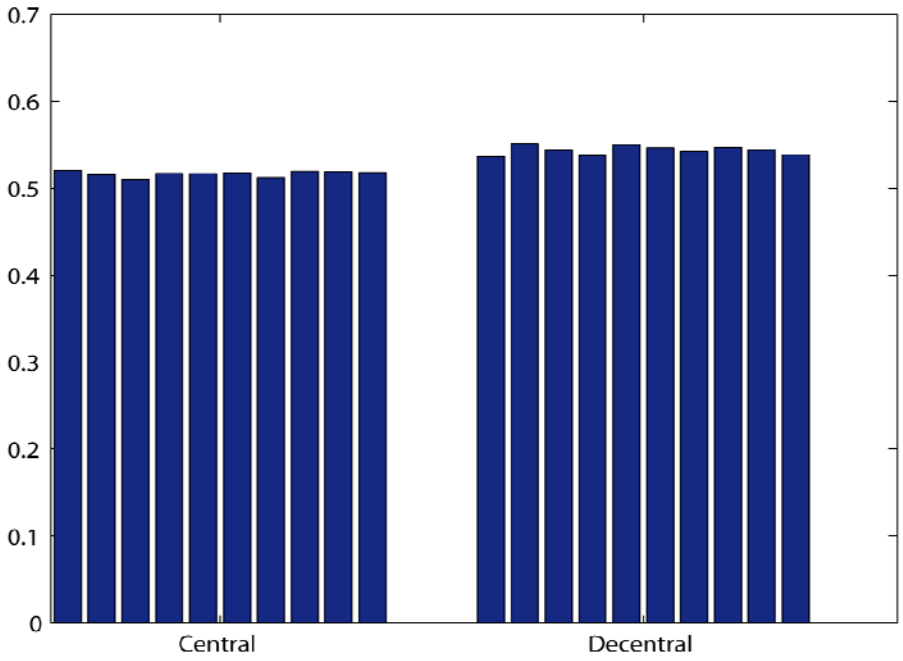

Figure 12: Final bar decentralized vs. centralized

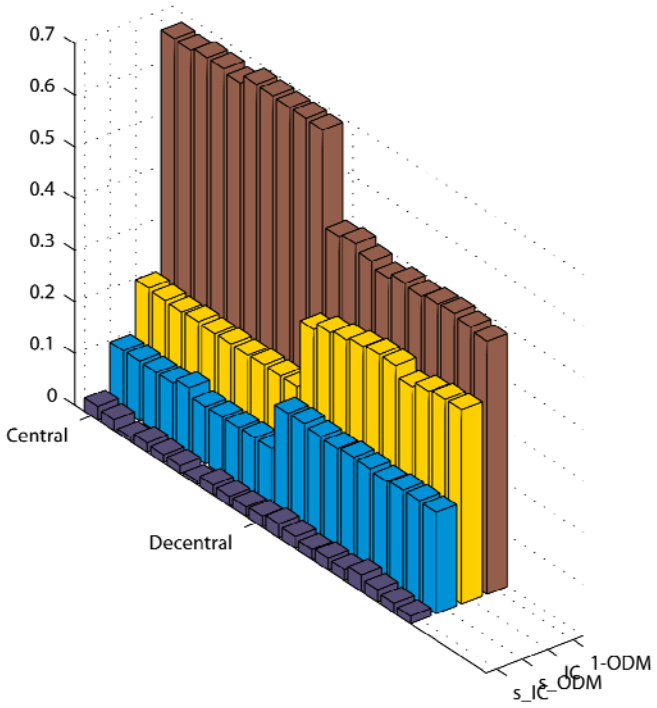

Figure 13: ODM and IC decentralized vs. centralized

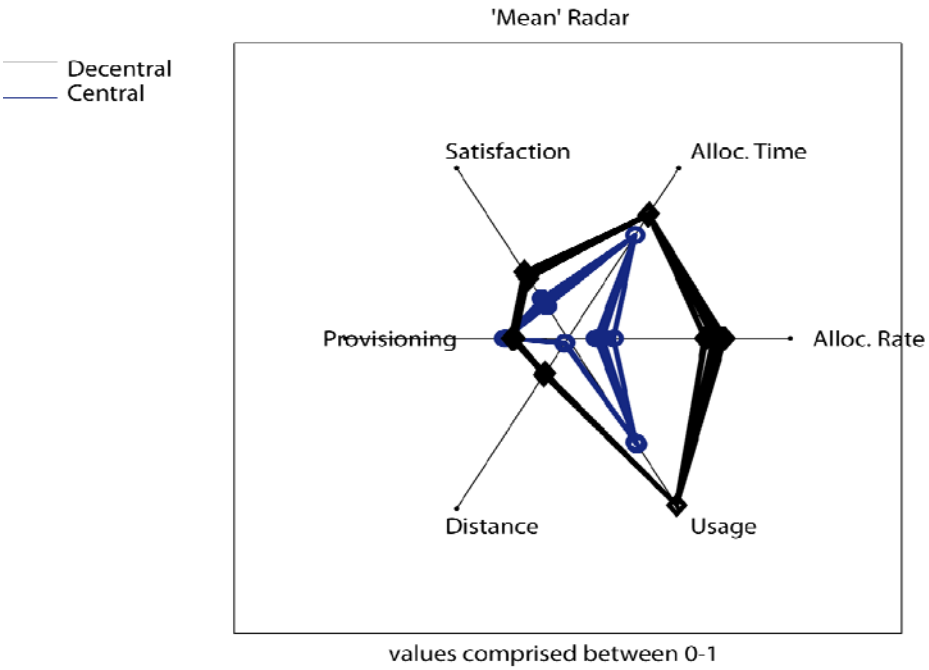

Figure 14: Spider mean centralized vs. decentralized

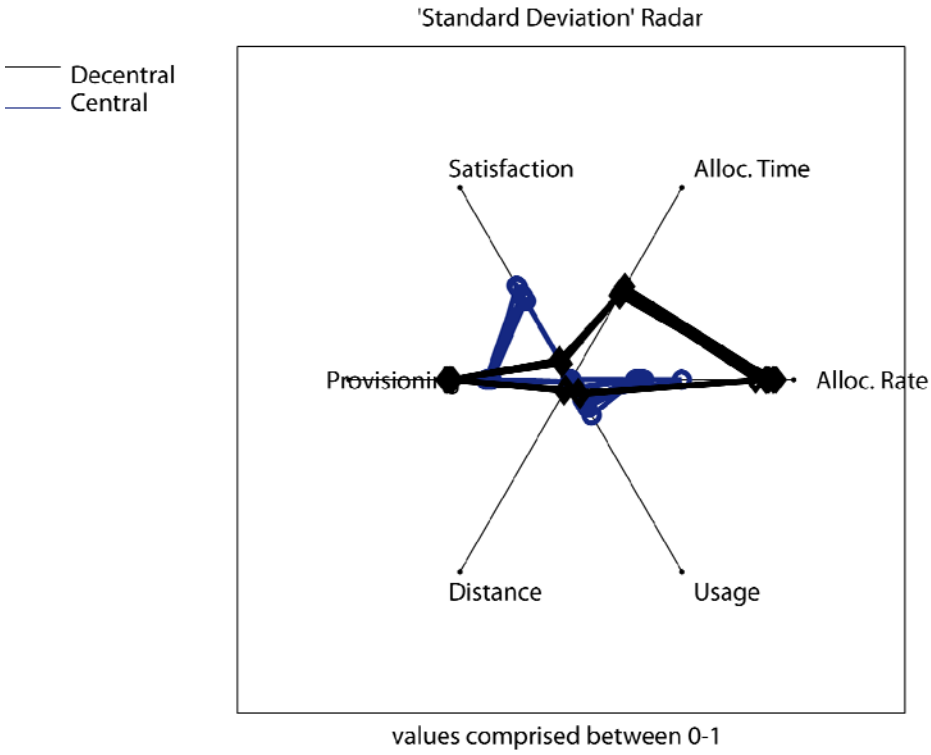

Figure 15: Standard deviation spider decentralized vs. centralized

3.6.4 Influence of hopcount on decentralized simulation results

This section analyzes the influence of the hopcount parameter on the decentralized simulation results. The parameter hopcount determines the range of call for proposal messages. The analysis was done based on the scenario 50A_10N. The hopcount parameter is evaluated with a value of zero, two and four. For each parameter setting, 10 simulation runs are executed.

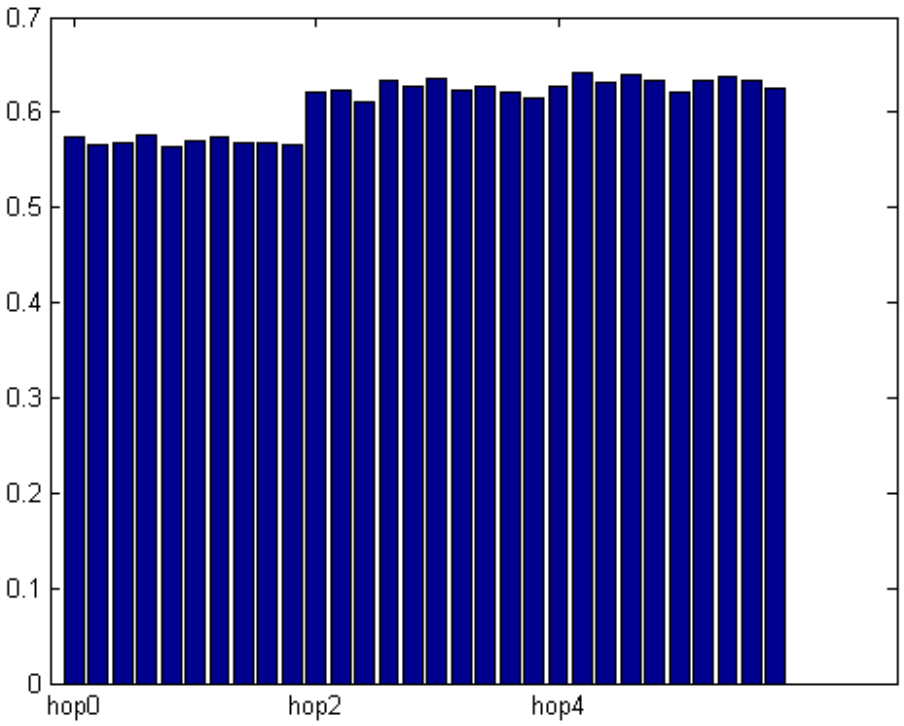

Figure 16: Final bar hopcount comparison

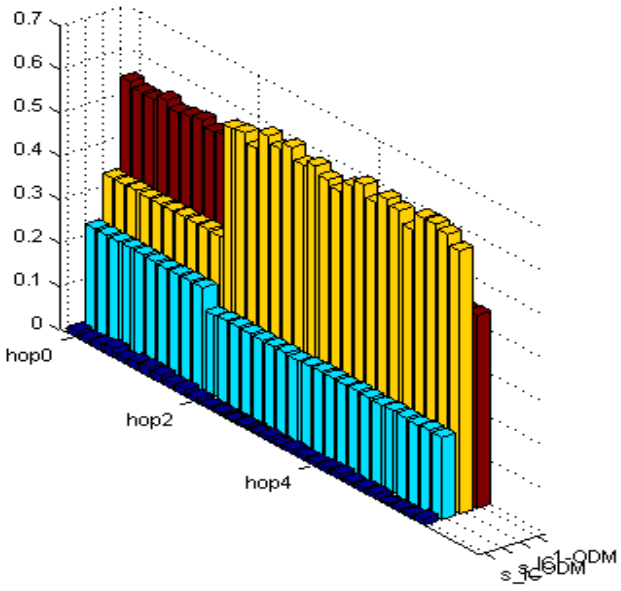

Figure 17: ODM and IC hopcout comparison

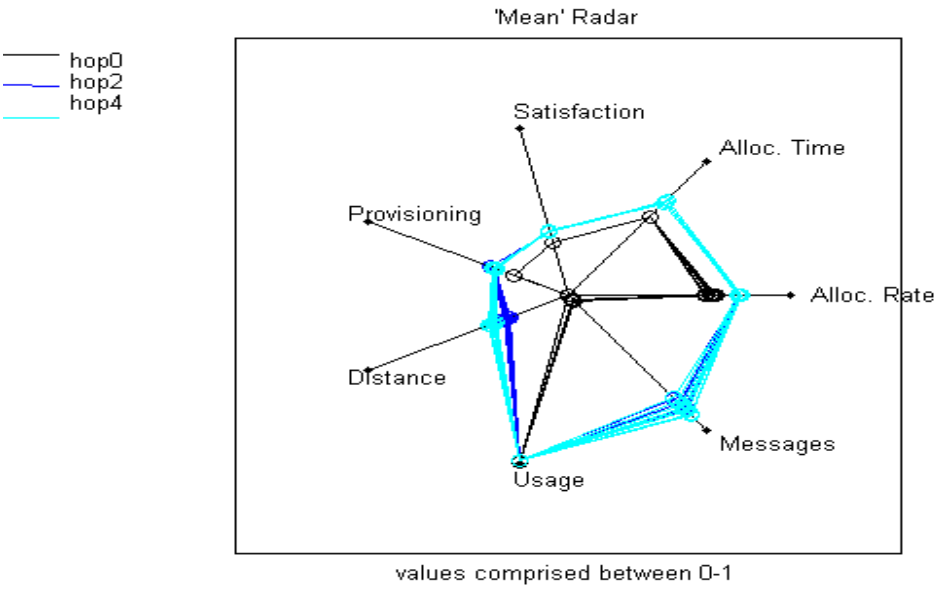

Figure 18: Mean spider hopcount comparison

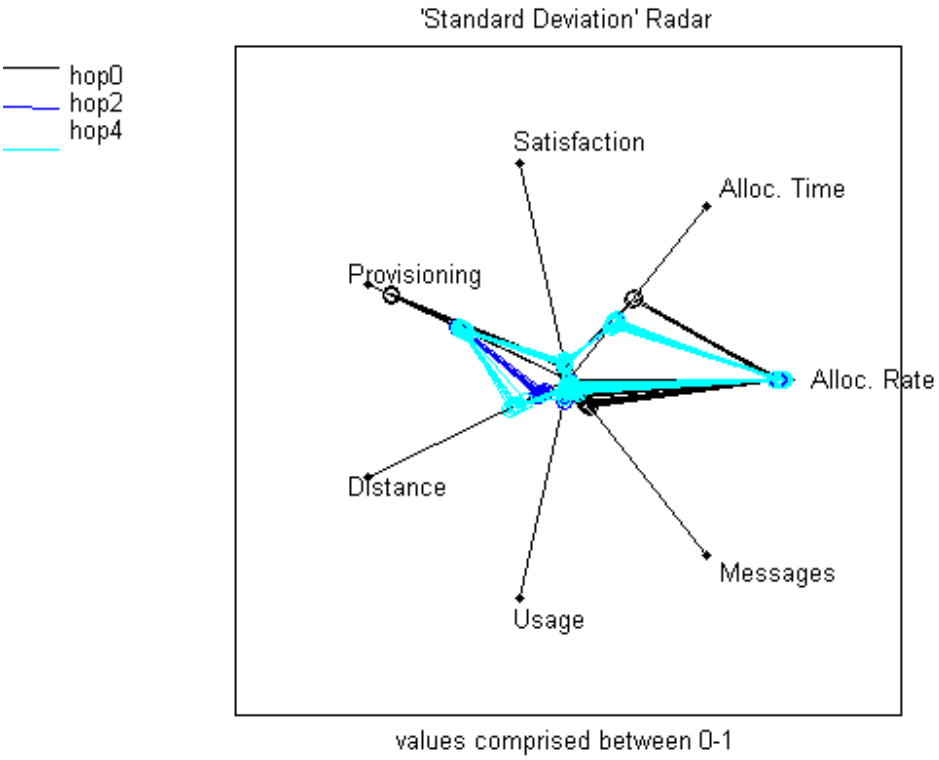

Figure 19: Standard deviation spider hopcount comparison

The Figure 16, Figure 17, Figure 18 and Figure 19 show the results for each hopcount setting. Figure 16 shows the final values for each parameter setting. The best values are

achieved if hopcount is set to zero. If hopcount is set to two or four the results differ only slightly. Figure 17 depicts IC and 1-ODM values for each experiment type. It can be observed that a raising hopcount causes higher mean IC costs. On the other hand, the mean 1-ODM value and its deviation are decreasing the higher the parameter hopcount was set. This can be explained by the graphs depicted in the Figure 18 and Figure 19. Figure 18 shows that the parameters influencing the 1-ODM value are raising only slightly for a higher hopcount. But the message parameter, which drives IC value, is rising by a large value. The standard deviations for the different parameters are almost stable (Figure 19). Small deviations can be observed for the parameters distance and allocation rate. The experiments show that raising the hopcount parameter lowers the final value. The gain that can be achieved by communicating to a bigger set of negotiation partners is lower than the costs arising to enable the communication. In a topology with high a density of agents, the hopcount parameter has to be set as low as possible.

3.6.5 Evaluation of the catallactic approach with failure swichted on

This experiment analyses the influence of message failure on system performance of the catallactic strategy. 300 agents were distributed over a topology with 500 nodes. The failure rate of each node increases from 0% up to 10%. Two different catallactic strategy variations are compared to each other.

Description	Configuration

complex service types and their basic service configuration	`cs1 bs3`
	`cs2 bs1 bs2`
	`cs3 bs1 bs4`

basic services and their requested resource bundle	`bs1 bs1 bronze r1 3 r2 3`
	`bs2 bs2 gold r4 2`
	`bs3 bs3 bronze r1 25 r3 10`
	`bs4 bs4 bronze r4 33 r5 25`

resource provider types and available resources for each type	`arb1 r1 50 r2 30 r3 30`
	`arb2 r4 50 r5 50`
	`arb3 r1 50 r3 44 r4 45`

Table 11. Service and resource supply and demand configuration.

The configuration of the service market encompasses three complex service types. The detailed configuration shows Table 11. Complex service cs1 requests only basic service bs3 while complex services cs2 and cs3 need two basic services (bs1 and bs2 for cs2, bs1 and bs4 for cs3) to fulfill their demand. The user demands are equally distributed to available complex services in the system. Every site, which hosts a complex service, is able to process cs1, cs2 and cs3 complex service user demands. On the service market, sellers offer four basic services, which are all requested by complex services. On the resource market, the basic services bs1 – bs4 request a resource bundle from the three available resource provider types arb1 – arb3. The resource providers use a dedicated resource model, which assigns the whole resource to one single basic service. The not

used resources are not available for another basic service. Using this resource model, bs1 and bs3 compete for resource provider arb1, bs2 and bs4 compete for resource provider arb2 and bs2 and bs3 compete for resource provider arb3.

In this scenario, the products on the service market are bs1 – bs4. The prices follow the price configuration of Table 12. The left side of the table presents the basic service price configuration; the right side of the table shows the corresponding price configuration of the resource market. Each basic service requests one specific resource product. The hard upper and lower limit prices for basic services on the resource market are 5 units above the price level of the corresponding limits of the resource products traded on the resource market. This forces resources to be cheaper than the services on the service market in general. But, there is still the possibility of a resource product to be more expensive than a basic service is able to pay. If a basic service bs1 sells his service for 55 price units and the resource bundle is traded with 70 resource units, a basic service bs1 fails in buying the resource bundle. For a detailed explanation of the individual properties, the reader is referred to deliverable D2.3.

Basic service price configuration	Resource price configuration
bs1.seller.minPrice = 55 bs1.seller.maxPrice = 65 bs1.buyer.minPrice = 55 bs1.buyer.maxPrice = 65 bs1.hard.lower.limit = 25 bs1.hard.upper.limit = 85 bs1.resource.itemids = r1r2_0	r1r2_0.seller.minPrice =50.0 r1r2_0.seller.maxPrice =60.0 r1r2_0.buyer.minPrice =50.0 r1r2_0.buyer.maxPrice =60.0 r1r2_0.hard.lower.limit =20.0 r1r2_0.hard.upper.limit =80.0 r1r2_0.baseunit.r1= 1 r1r2_0.baseunit.r2= 1 r1r2_0.resourceids = r1 r2
bs2.seller.minPrice = 30 bs2.seller.maxPrice = 35 bs2.buyer.minPrice = 30 bs2.buyer.maxPrice = 35 bs2.hard.lower.limit = 15 bs2.hard.upper.limit = 45 bs2.resource.itemids = r4_0	r4_0.seller.minPrice =25.0 r4_0.seller.maxPrice =30.0 r4_0.buyer.minPrice =25.0 r4_0.buyer.maxPrice =30.0 r4_0.hard.lower.limit =10.0 r4_0.hard.upper.limit =40.0 r4_0.baseunit.r4= 1 r4_0.resourceids = r4
bs3.seller.minPrice = 55 bs3.seller.maxPrice = 65 bs3.buyer.minPrice = 55 bs3.buyer.maxPrice = 65 bs3.hard.lower.limit = 25 bs3.hard.upper.limit = 85 bs3.resource.itemids = r1r3_0	r1r3_0.seller.minPrice =50.0 r1r3_0.seller.maxPrice =60.0 r1r3_0.buyer.minPrice =50.0 r1r3_0.buyer.maxPrice =60.0 r1r3_0.hard.lower.limit =20.0 r1r3_0.hard.upper.limit =80.0 r1r3_0.baseunit.r1= 1 r1r3_0.baseunit.r3= 1 r1r3_0.resourceids = r1 r3
bs4.seller.minPrice = 55 bs4.seller.maxPrice = 65 bs4.buyer.minPrice = 55 bs4.buyer.maxPrice = 65 bs4.hard.lower.limit = 25 bs4.hard.upper.limit = 85	r4r5_0.seller.minPrice =50.0 r4r5_0.seller.maxPrice =60.0 r4r5_0.buyer.minPrice =50.0 r4r5_0.buyer.maxPrice =60.0 r4r5_0.hard.lower.limit =20.0 r4r5_0.hard.upper.limit =80.0

`bs4.resource.itemids = r4r5_0`	`r4r5_0.baseunit.r4= 1`
	`r4r5_0.baseunit.r5= 1`
	`r4r5_0.resourceids = r4 r5`

Table 12. Initial price configuration for services and resource bundles.

Next to the scenario configuration, the main simulator configuration integrates the configuration described above and sets the simulation parameters. A user submits 10000 complex requests to the simulation scenario. Each complex service type is requested equally. The complex service dispatcher randomly selects an inbox queue of complex service instance hosted on a Grid site. The delay between each complex service request is set to 1000ms. The delay remains constant during the simulation run. The execution time of each basic service is 1000ms. A constant execution time disables the effects of service execution times on the resource allocation approach. Both markets are connected to each other. This means, the agreement price of a basic service seller equals the budget of a resource buyer. As already mentioned, the resource providers use a dedicated resource model. Co-allocation of resource on the resource market is switched off. A buyer uses a fifo policy for selecting a seller to negotiate with. The fastest answering seller is selected. The hop count for all broadcast messages is set to 3 hops. A discovery timeout of 500ms limits the waiting time for reaching a proposal. If a negotiation partner does not answer at all, a negotiation timeout of 2500ms for each market resets the negotiation. The size of each message is set to 2 kByte which will lead to low delays on the network.

Strategy 1	Strategy 2
maturityThreshold = 5	maturityThreshold = 5
courterThreshold = 20	courterThreshold = 20
crossoverProbability = 0.20	crossoverProbability = 0.20
mutationProbability = 0.7	mutationProbability = 0.05
ringSize = 10000	ringSize = 10000
crossOverSelectionModel = 0	crossOverSelectionModel = 0
gaussWidth = 0.1 min = 0.001 max = 0.999	gaussWidth = 0.01 min = 0.001 max = 0.999
genotype.randomize = yes genotype.acquisitiveness = [0.4, 0.8] genotype.satisfaction = [0.4, 0.8] genotype.priceStep = [0.1,0.4] genotype.priceNext = [0.1,0.6] genotype.weightMemory = [0.3,0.7]	genotype.randomize = no genotype.acquisitiveness = 0.05 genotype.satisfaction = 0.99 genotype.priceStep = 0.5 genotype.priceNext = 0.05 genotype.weightMemory = 0.9

Table 13. Two strategy configurations for the failure scenario analysis.

Two different learning setups were compared to each other for the defined failure scenarios whose configuration lists Table 13. The characteristic of the first strategy is a randomized initial behavior and fast adaption to new behavior. A random number is drawn between the given interval bounds for each gene of the agent. This assures a

diverse trading behavior of the agents. Additionally, a mutation probability of 0.7 forces the agent to mutate 70% of the 5 genes in every learning step. A large Gaussian width causes adaptations steps up to 10% of the current gene value. Together with the high mutation rate, the agents have the possibility to move very fast away from an initial bad trading performance.

The strategy 2 does not randomize the initial trading behavior. Every agent uses the given genotype as a start behavior. The characteristic of this behavior is a high satisfaction value together with a high price step value which leads to only a few negotiations round and low cancellation of negotiations. A low acquisitiveness forces the agent to make concessions even if the agent doesn't make profit any more. The genes of the genotype are quite stable, only with a probability of 5% a gene is mutated with at maximum 1% of the gene's value (Gaussian width).

Using this simulation and strategy configuration, Table 14 gives an overview of the simulation runs. 10 simulations with two strategies and failure settings are examined. The following plots print only the experiment id due to amount of space. The table helps to map the results to the experiment setting.

Experiment Id	Parameter setup
1189323882133	Strategy 1, 0% failure
1189326383709	Strategy 1, 1% failure
1189327402864	Strategy 1, 2% failure
1189328268368	Strategy 1, 5% failure
1189330779897	Strategy 1, 10% failure
1189333572714	Strategy 2, 0% failure
1189337074555	Strategy 2, 1% failure
1189344749997	Strategy 2, 2% failure
1189347224464	Strategy 2, 5% failure
1189348828710	Strategy 2, 10% failure

Table 14. Mapping of experiment ids to parameter setup.

Figure 20 depicts the overall performance of the catallactic allocation approach. The strategy shows less loss from the optiomal allocation performance in both 0% failure simulation runs. The final index value is about 0.1 units better in the second strategy than in the first strategy. The second strategy shows a better inverse on demand availability (1-ODM) than the first strategy. But, the infrastructure costs remain almost the same. As expected, the introduction of failure to the system increases the non availibility and leads to more loss of the system. Additionally, the infrastrubure costs increase. The standard deviations of both indexes IC and ODM equal in both strategy settings.

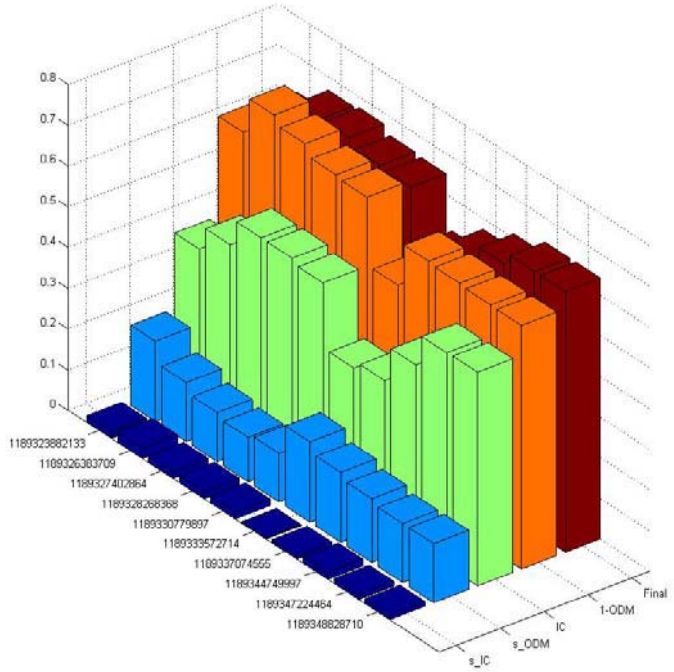

Figure 20. Bar graph for 10 simulations runs and different scenario setup. The simulations runs are compared with the On Demand Availability (ODM) and Infrastructure Cost (IC) index which are used to compute the final loss function (Final).

A more detailled analysis enables the mean and stadard deviation radar plot in Figure 21 and Figure 22. Seven metric values are aggregated on agent population level and normalized between the interval 0 and 1. Each colored line equals an experiment run. The experiment with the yellow line (0% failure and strategy 1) shows the the best average satisfaction value with low standard deviation. Beside the best allocation time, this simulation run has second best allocation rate. A very high number of messages are needed to reach this performance. All agents in the system show high usage. This means, the agents show low idle times. There are two reasons for low idle times: first the agent spends lots of time in negotiations and deliverance of services or the agents are blocked until simulation end due to lost unblocking messages. Additionally, the strategy 1 applied to this scenario selects trading partners with low distance. The provisioning time is worse in failure scenarios. This occurs due to the low number of obersvation in these scenarios. All other simulation runs spend more time on service provisioning.

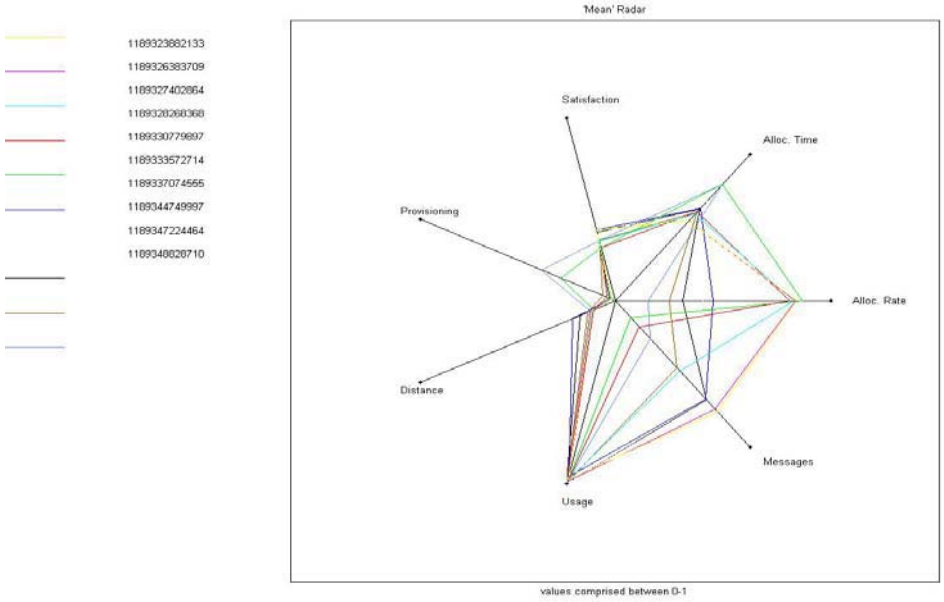

Figure 21. Spider plot for 10 simulations runs.

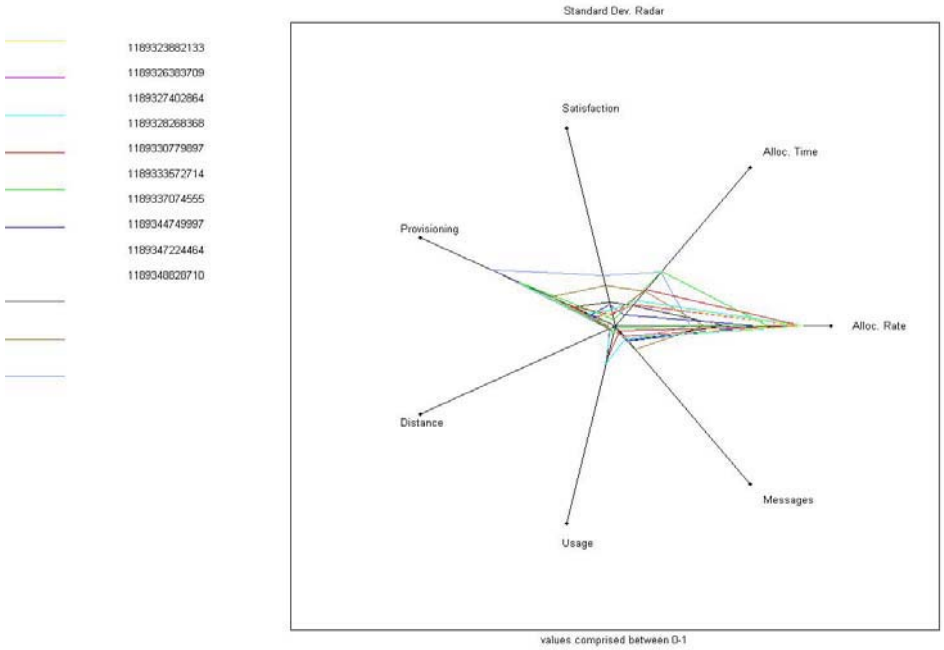

Figure 22. Spider plot for 10 simulation runs**.**

Table 15 lists the number of observations used in the scenario for index computations. Strategy 2 with the predefined genotype outperforms strategy 2 with the randomized genotype initialization. Increasing the failure rate rapidly decreases the number of observation. Only a few of the 10000 requests are successfully completed. The current catallactic strategy implementation is very sensitive to message failures. There is also a difference between the numbers of agent involved in trades. This number is decreases in strategy 1 faster than in strategy 2.

Experiment Id	Observation
1189323882133	BSA_buyer: 43, BSA_seller: 79, RSA: 34, CSA: 54 1194 observations (accepts SM + RM)
1189326383709	BSA_buyer: 38, BSA_seller: 77, RSA: 27, CSA: 52 356 observations (accepts SM + RM)
1189327402864	BSA_buyer: 8, BSA_seller: 54, RSA: 8, CSA: 40 78 observations (accepts SM + RM)
1189328268368	BSA_buyer: 1, BSA_seller: 40, RSA: 1, CSA: 28 51 observations (accepts SM + RM)
1189330779897	BSA_buyer: 8, BSA_seller: 38, RSA: 8, CSA: 31 46 observations (accepts SM + RM)
1189333572714	BSA_buyer: 65, BSA_seller: 86, RSA: 53, CSA: 54 14518 observations (accepts SM + RM)
1189337074555	BSA_buyer: 67, BSA_seller: 90, RSA: 54, CSA: 54 1911 observations (accepts SM + RM)
1189344749997	BSA_buyer: 64, BSA_seller: 86, RSA: 50, CSA: 51 571 observations (accepts SM + RM)
1189347224464	BSA_buyer: 55, BSA_seller: 77, RSA: 44, CSA: 49 242 observations (accepts SM + RM)
1189348828710	BSA_buyer: 53, BSA_seller: 77, RSA: 46, CSA: 49 220 observations (accepts SM + RM)

Table 15. Observation and involved agents for index computation

Summarizing the failure experiments, the catallactic strategy shows a high messaging vulnerability in the analyzed scenarios. The reason is the high number of messages which have to be transferred until an agreement is closed. The final system loss increases with the increasing unavailability of the agents and their services. But, the numbers of

observation have to be taken into accounts, which prevent higher loss measures of the system. The infrastructure costs also increase with higher failure rates because agents wait for the trading partners to answer until a predefined timeout. In case of no failure, the standard deviations are higher than in case of message failure. Agents trade to reach an agreement with very different success. This is the reason for a high allocation rate deviation in the failure free simulation runs. The main reason for the different success is the one shot policy of the agents (agents have only one try to reach an agreement) and no parallel negotiations supported.

3.6.6 Decentralized approach and the learning algoithm

This experiment analysis the co-evolutionary algorithm used in the catallactic strategy for adapting the strategy's genotype. The genotype, the price estimation and the fitness evolvement is measured in a large scale scenario. The automated scenario generator is used for generating this scenario. The scenario topology follows the Waxman model of Brite with 2000 nodes randomly distributed. Table 16 shows the scenario generator configuration. The bandwidth of the links was set to a fixed value. This will almost exclude any bandwidth influence on the simulation results. Table 16 gives an overview of the CATNETS service and resource market scenario setting for the automated scenario generator. Keeping the complexity low, the scenario generator produces a set of three resource providers with a resource bundle size of at maximum 3 resource items and a maximum quantity of 100 resource units. The set of basic services and complex service is also limited to 3 different types. Additionally, the workflow length of a complex service is limited to 3 basic services. The topology is assumed to be stable, no failures occur. The automated scenario generator places 2000 agents on the topology with 20 % complex service and 40 % basic service and resource service using a uniform distribution to map the number of agents to service and resource types. The last two configuration entries BSTable and ARBTable display the mapping of how many instances of each agent type are distributed on the network nodes.

Scenario generator parameter	Value
`#Resources Number`	`ResNum = 3`
`#Resource Max Quantity`	`ResMaxQuantity = 100`
`#Available Resource Bundle Number`	`ARBNumber = 3`
`#Available Resource Bundle Max Number`	`ARBMaxResNum = 3`
`#Basic Service Number`	`BSNumber = 3`
`#Complex Service Number`	`CSNumber = 3`
`#Complex Service Max Cardinality`	`CSMaxCardinality = 3`
`#Node Min Failure probability`	`FailProbMin = 0.0`
`#Node Max Failure probability`	`FailProbMax = 0.0`
`#Quality Number`	`QualityNumber = 4`
`#Quality Level`	`Quality0 = platinum, Quality1 = gold, Quality2 = silver, Quality3 = bronze`
`#0 Centralized; 1 Catallactic`	`allocationMechanism = 1`
`#Agents Number`	`agentsNum = 2000`

#CS Schedule 0= All, 1= random set	csSchedule = 0
#Agent share in percent	csaPercentage = 20, bsaPercentage = 40, raPercentage = 40
#Probability distribution (0 = uniform)	csaDistrProb = 0, bsaDistrProb = 0, raDistrProb = 0
#BSTable	bs1 = 33.0, bs2 = 33.0, bs3 = 33.0
#ARBTable	arb1 = 33.0, arb2 = 33.0, arb3 = 33.0

Table 16. Scenario generator parameters for generating complex service, basic service and available resource bundle types and their distribution over the network topology using probability distributions.

The automated scenario generator creates the scenario presented in Table 17. There is a supply of three different basic service types and a demand of three different basic service sequences of the complex services. The resource market offers three resource bundles which are asked by basic services.

Description	Configuration
complex service types and their basic service configuration	cs1 bs2 cs2 bs1 bs3 bs2 cs3 bs2 bs3
basic services and their requested resource bundle	bs1 bs1 bronze r1 20 r3 3 bs2 bs2 bronze r2 52 bs3 bs3 bronze r2 6 r3 1
resource provider types and available resources for each type	arb1 r1 21 r3 10 arb2 r2 25 r3 1 arb3 r2 59

Table 17. Generated scenario configuration with different workflow lengths for complex services and three different complex service, basic service and resource types.

The price configuration is similar to the price configuration described in the last section. The prices help to explain the following analysis of the price estimation of the agents presented later in this section. The hard upper and lower limit price help to identify the price ranges of the traded products.

Basic service price configuration	Resource price configuration
bs1.seller.minPrice = 55 bs1.seller.maxPrice = 65 bs1.buyer.minPrice = 55 bs1.buyer.maxPrice = 65 bs1.hard.lower.limit = 25 bs1.hard.upper.limit = 85 bs1.resource.itemids = r1r3	r1r3.seller.minPrice =50.0 r1r3.seller.maxPrice =60.0 r1r3.buyer.minPrice =50.0 r1r3.buyer.maxPrice =60.0 r1r3.hard.lower.limit =20.0 r1r3.hard.upper.limit =80.0 r1r3.baseunit.r1= 20 r1r3.baseunit.r3= 3 r1r3.resourceids = r1 r3
bs2.seller.minPrice = 30 bs2.seller.maxPrice = 35 bs2.buyer.minPrice = 30 bs2.buyer.maxPrice = 35 bs2.hard.lower.limit = 15 bs2.hard.upper.limit = 45	r2.seller.minPrice =25.0 r2.seller.maxPrice =35.0 r2.buyer.minPrice =25.0 r2.buyer.maxPrice =35.0 r2.hard.lower.limit =20.0 r2.hard.upper.limit =40.0

bs2.resource.itemids = r2	r2.baseunit.r2= 52 r2.resourceids = r2
bs3.seller.minPrice = 55 bs3.seller.maxPrice = 65 bs3.buyer.minPrice = 55 bs3.buyer.maxPrice = 65 bs3.hard.lower.limit = 25 bs3.hard.upper.limit = 85 bs3.resource.itemids = r2r3	r2r3.seller.minPrice =50.0 r2r3.seller.maxPrice =60.0 r2r3.buyer.minPrice =50.0 r2r3.buyer.maxPrice =60.0 r2r3.hard.lower.limit =20.0 r2r3.hard.upper.limit =80.0 r2r3.baseunit.r2= 6 r2r3.baseunit.r3= 1 r2r3.resourceids = r2 r3

Table 18. Price configuration for basic services and resource bundles.

The simulation run is started with 100000 complex service requests with a delay of 1000 ms between. The maximum queue size is increased to 20000 to ensure there is enough capacity to store all requests, if the system is not able to fulfill the current request faster than the incoming requests. The execution time of basic services is set to 1000 milliseconds. Both, the service and the resource market are connected. The service seller's income is his budget on the resource market. The resource providers use a didcated resource model. The buyers select the first incoming proposal for the succeeding bilateral negotiation. Co-allocation is switched off. The call-for-proposal and the plumage broadcast is limited to 3 hops. The discovery timeout is set to 500 ms. The simulation run uses the advanced Grid time model.

Strategy 1
maturityThreshold = 5
courterThreshold = 20
crossoverProbability = 0.20
mutationProbability = 0.7
ringSize = 10000
crossOverSelectionModel = 0
gaussWidth = 0.01 min = 0.001 max = 0.999
genotype.randomize = yes genotype.acquisitiveness = [0.4, 0.8] genotype.satisfaction = [0.4, 0.8] genotype.priceStep = [0.1,0.4] genotype.priceNext = [0.1,0.6] genotype.weightMemory = [0.3,0.7]

Table 19. Strategy configuration for the simulated scenario; the initial genotype is randomized between the given interval limits.

Table 19 presents the selected starting configuration of the strategy and the learning algorithm. The initial genotype of each agent is randomized at the beginning between the given interval limits. A high mutation probability and a low Gaussian width enable the agent to adapt their genes often and in small steps. They wait for 5 generations until they broadcast their plumage and perform a crossover when they have received 20 plumages.

A pair of agents for each agent role was chosen for analysis. Figure 23 and Figure 24 show two pairs of complex service agents. The left plot displays the genotype evolution, the plot in the middle the price estimations and right plot the fitness evolution for 200 and 500 observations. An observation is a successful trade between a complex service agent and a basic service agent. In Figure 23, both agents are very successful in their trades. Their fitness increases from 0 at the beginning to a peak around 10 to a quite stable fitness value. One reason for this high fitness value is a low price step. Together with a decreasing satisfaction value, the agents follow a strategy of fast agreement with only low price concessions. The agent with a higher price concession rate (acquisitiveness) is able to make more profit. A weighting memory value of 0.6 seems a good parameter taking old agreement prices into account for new price range estimations.

The price estimation plot shows for both agents of Figure 23 basic service agent trades at different estimated market price level. Both trade at least two different basic services. The market price estimation is more stable for the upper agent than the lower one. The means, the upper complex service agent gets enough offers from basic service. He hasn't to increase his price estimations to a possible scarceness of basic services. The lower agent has to increase his price estimations to be able to contract basic services.

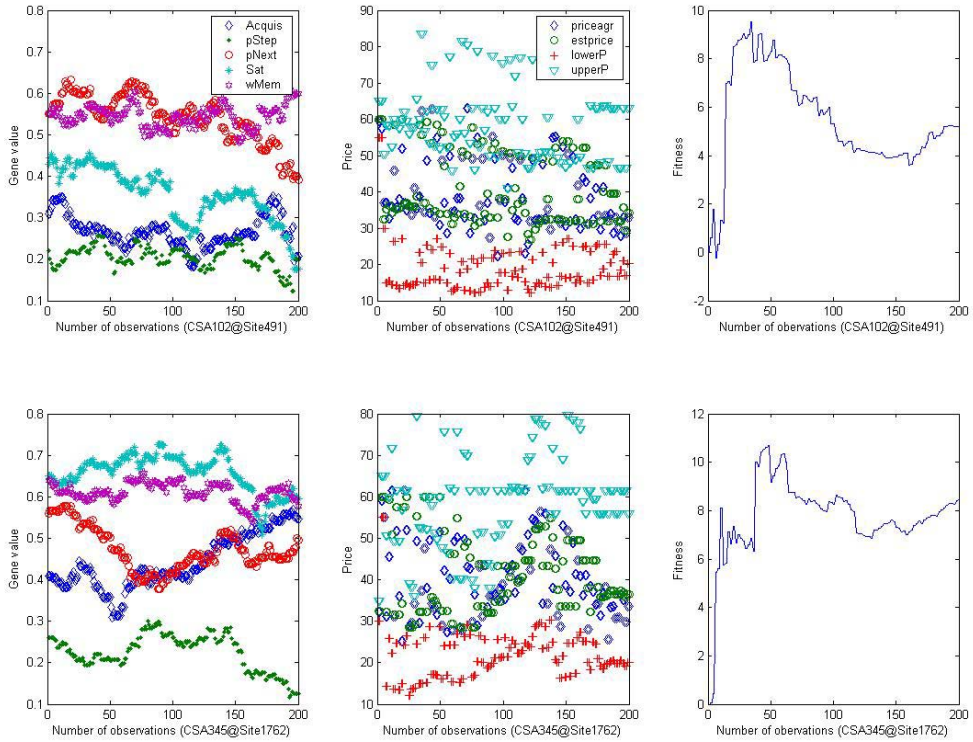

Figure 23. A pair of successful complex service agents in the simulated scenario; the left graph displays the genotype evolution, the plot in the middle the price estimations and right plot the fitness evolution for 200 observations.

In Figure 24, a second set of complex services is selected. This set isn't as successful as the first set in estimating the market price but engages in more trades than the first set. The set uses a higher satisfaction value than the first set, which leads to more negotiation round and a higher possibility of reaching an agreement. The upper agent is more successful in making some large concession steps than the lower agent. A low acquisitiveness value between 0 and 5 % still enables the agent to reach agreements but with only with negative fitness.

Summarizing up, successful strategies of service sellers follow a strategy with a low concession rate together with a high concession step or a high concession rate with lower concession steps. The higher the satisfaction value, the more successful trades are possible.

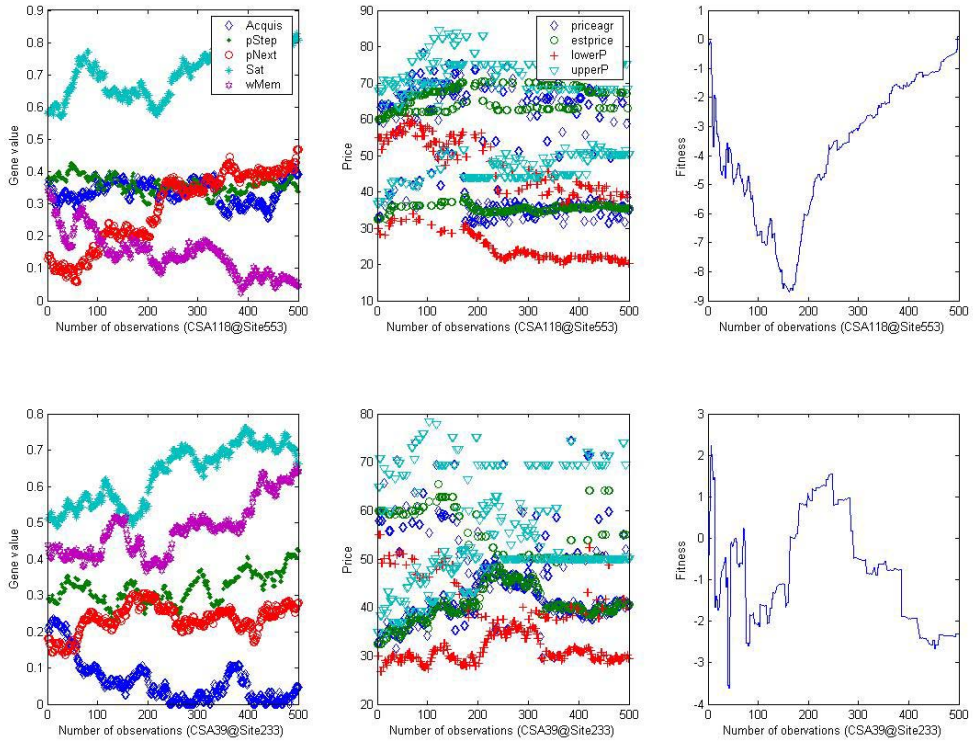

Figure 24. A pair of unsuccessful complex service agents in simulated scenario; the left graph displays the genotype evolution, the plot in the middle the price estimations and right plot the fitness evolution for 500 observations.

The basic services have two roles on in the CATNETS scenario: a seller role on the service market and a buyer role on the resource market. Figure 25 presents a set of selected basic service sellers for 300 observations. Both agents show a positive fitness, which indicates a successful strategy. They follow the strategy of many negotiation rounds together with a low concession rate and high concession steps. But, their market price estimation differs a lot. The upper basic service seller is at the beginning not successful. He has to lower his price estimations. After a while, he gets more and more selected and increases the prices for his service. The lower agent is successfully at the beginning and has to lower his prices step by step because he did not succeed in negotiations. As a result of this, his fitness decreases over time.

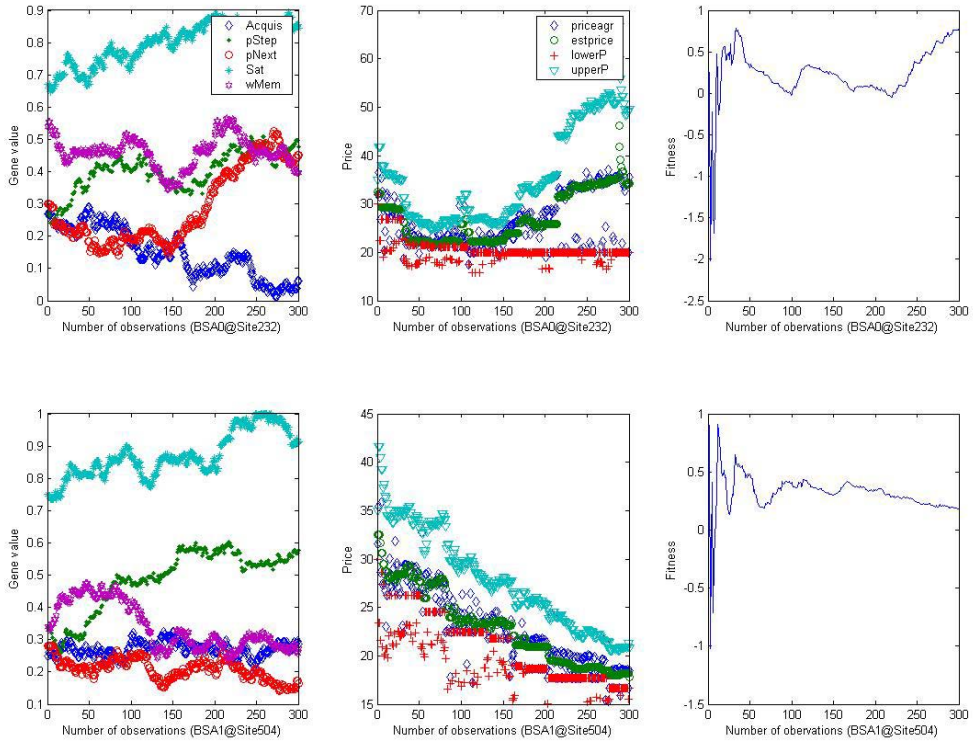

Figure 25. A pair of basic service sellers in the simulation scenario; the left graph displays the genotype evolution, the plot in the middle the price estimations and right plot the fitness evolution for 300 observations.

Figure 26 and Figure 27 plot a set of basic service buyer agents and resource agents. In general, the number of observation decreases on the service market because resource negotiations are started only after successful service market agreements. One basic service buyer agent with 60 and one agent with 150 observations present Figure 26. Both agents depend on the results of the service market seller agents. This agreement price is used to adapt the price ranges. Comparing to the price range adaptation before, the upper and lower bounds show higher variability. The market price estimations for resource products are fairly stable. The lower agent shows a successful strategy, low price steps together with a high concession and satisfaction rate. The upper agent is not able to reach positive fitness values because he isn't able to reach his current price estimation due to a high concession step. In the current simulation scenario configuration, it is not possible to initialize the resource market with a different strategy setup. A faster adaptation of the genotype on the resource market could help to improve the fitness results.

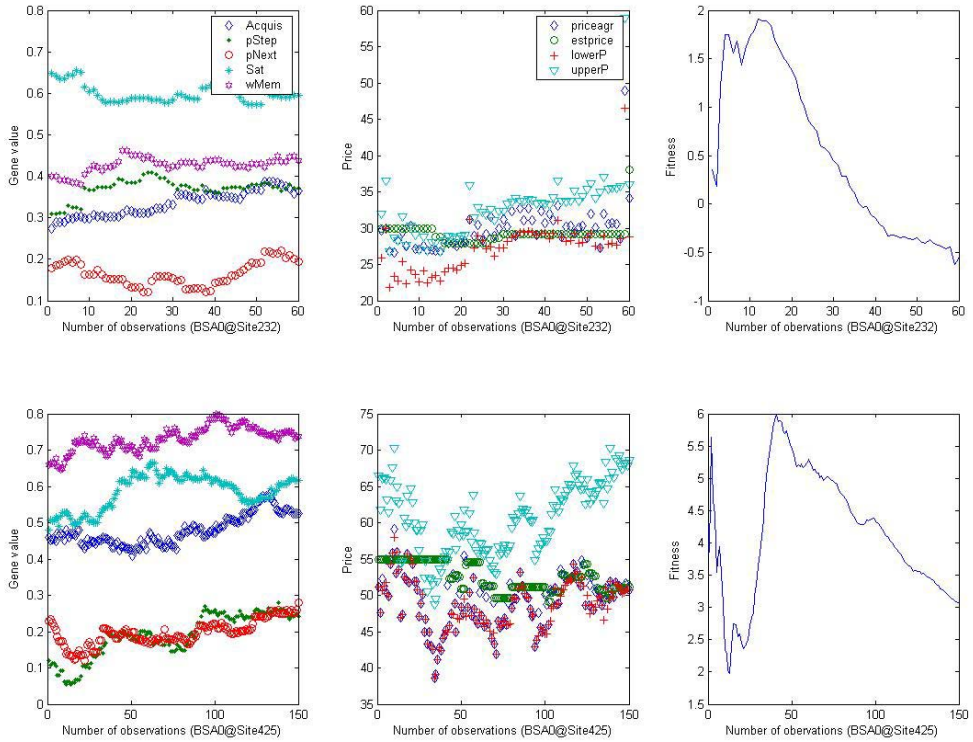

Figure 26. A pair of basic service buyers; the left graph displays the genotype evolution, the plot in the middle the price estimations and right plot the fitness evolution for 150 observations.

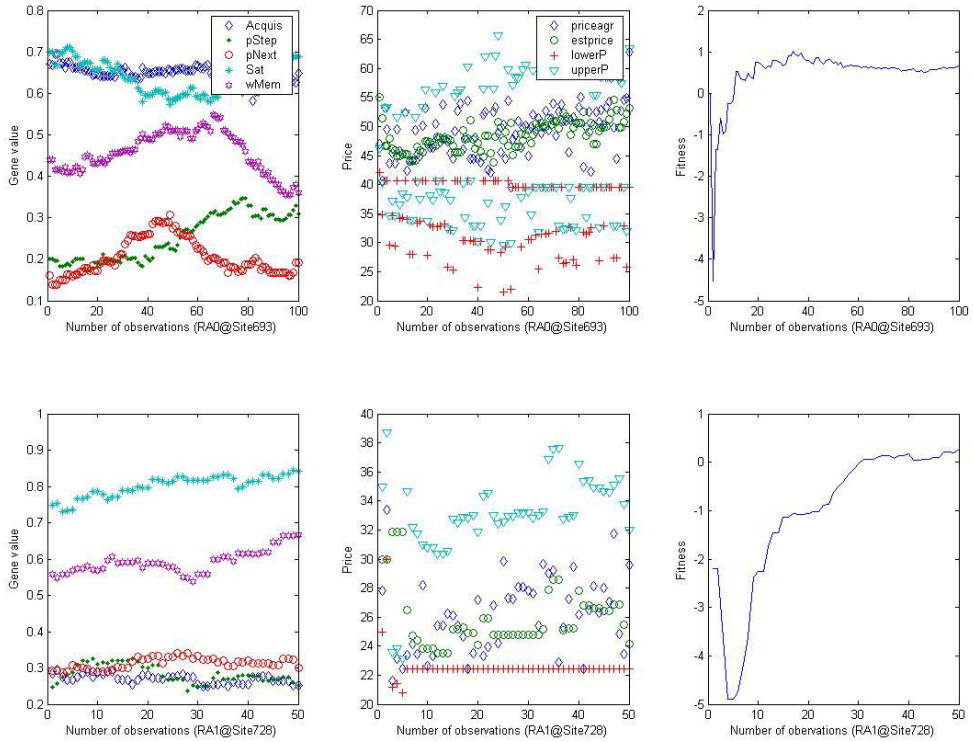

Figure 27. A pair of resource agents; the left graph displays the genotype evolution, the plot in the middle the price estimations and right plot the fitness evolution for 50 observations.

Finally, Figure 27 depicts two selected resource agents. Both agents show similar results as before. Only a few observations are recorded for the resource agents because there is no scarceness of resources. From the total number of 2000 agents, 40% are resource agent, which are requested by 40% basic service agents. The basic service agents have the possibility to choose between several resource agents in between their 3 hop broadcast limit. The fitness of both agents reaches a stable point slightly above 0. The agent strategy with higher acquisitiveness and a higher price step is more successful according the successful trades. The upper agent is able to trade several resource products because he has several distinct price intervals whereas the lower agent only offers one resource product. This also helps the upper agent to adapt his genotype more often to current situations.

3.6.7 Influence of bandwidth on the catallactic approach

This experiment analyses the effect of varying network bandwidth on the catallactic allocation approach. Four different bandwidth scenarios are compared to each other. In each scenario, 200 agents (66 CSA, 67 BSA, and 67 RSA) and a topology with 200 nodes

are simulated. The agents are distributed on the nodes using a uniform distribution. Keeping the scenario simple, there is only one complex service type which request one instance of a basic service bs1. A basic service bs1 requests one unit of resource service r1. The number of agents equals the number of service and resource instances. Table 20 shows the scenario configuration.

Description	Configuration
complex service types and their basic service configuration	`cs1 bs1`
basic services and their requested resource bundle	`bs1 bs1 bronze r1 1`
resource provider types and available resources for each type	`arb1 r1 1`

Table 20: Simple scenario configuration with a 1 to 1 to 1 mapping between complex services, basic services and resources

Depending on the service configuration, the price configuration also simplifies. Only prices for one basic service and one resource product have to be initialized. The large price interval limits and the same initial price interval for buyers and seller (see Table 21) will isolate effects of the price configuration on the simulation runs.

Basic service price configuration	Resource price configuration

`bs1.seller.minPrice = 100`
`bs1.seller.maxPrice = 160`
`bs1.buyer.minPrice = 100`
`bs1.buyer.maxPrice = 160`
`bs1.hard.lower.limit = 80`
`bs1.hard.upper.limit = 200`
 	
`bs1.resource.itemids = r1`
	`r1.resourceids = r1`
	`r1.baseunit.r1 = 1`
	`r1.seller.minPrice = 100`
	`r1.seller.maxPrice = 160`
	`r1.buyer.minPrice = 100`
	`r1.buyer.maxPrice = 160`
	`r1.hard.lower.limit = 80`
	`r1.hard.upper.limit = 200`

Table 21: Price configuration for basic service bs1 and resource bundle r1

The strategy and learning setup are presented in Table 22. The configuration is similar to the configurations of the simulation runs in the previous sections. Differences from previous strategy setups are the low mutation rate of 0.05 together with a low Gaussian width of 0.01. This keeps the change of the initial genotype fairly stable for all agents. The initial step size of 0.3 enables fast agreements between the negotiation partners. Again, this setup selection intends to isolate the effects of the agent's trading performance from the different bandwidth scenarios.

Strategy 1
maturityThreshold = 5
courterThreshold = 20
crossoverProbability = 0.20
mutationProbability = 0.05
ringSize = 10000
crossOverSelectionModel = 0

gaussWidth = 0.01 min = 0.001 max = 0.999
genotype.randomize = no genotype.acquisitiveness = 0.05 genotype.satisfaction = 0.99 genotype.priceStep = 0.3 genotype.priceNext = 0.05 genotype.weightMemory = 0.9

Table 22: Strategy configuration for the simulated scenario

During a simulation run, 1000 complex service requests are submitted to the system with a delay of 2000 milliseconds. A basic service execution guarantees low input queues because the system is able to process the requests fast. Again, both markets are connected to each other and a dedicated resource model is used. A buyer selects its proposals using the best price selection policy. For all simulation runs, the hop counters for broadcast messages are set to 4 hops. The discovery timeout is 500 milliseconds and the negotiation timeout 10000 milliseconds.

Table 23 lists the message size and bandwidth parameters. In the first simulation run, it is assumed a message size of 0 which is interpreted by the simulator as unlimited bandwidth. The messages are transported to the receiver without any latency. The second simulation run uses a fixed bandwidth of 100 and a message size of 10 whereas the third run uses a uniform distribution to assign a bandwidth between 100 and 1000 to the links of the network. Finally, the last experiment uses a fixed bandwidth of 1000 keeping the message size the same.

Experiment Id	Parameter setup
1190403910890	message.size = 0; bandwidth = unlimited
1190404122296	message.size = 10; bandwidth = 100
1190404999250	message.size = 10; bandwidth = [100, 1000] uniformly distributed
1190405643250	message.size = 10; bandwidth = 1000

Table 23: Mapping of experiment ids to parameter setup

For each experiment, 10 simulations runs are performed. One characteristic simulation run is selected for final analysis in Figure 28. The best final social utility achieves the experiment with unlimited bandwidth and no message delay. The instantaneous message delivery leads to high infrastructure costs in terms of number of messages and increased waiting times due to the blocking of agents during the discovery phase. But, the low distance between the trading partners and high agent satisfaction (see Figure 29) lead to high on demand availability. The 100 bandwidth scenario shows about 25 % worse system performance. Both infrastructure costs and inverse on demand availability increase. Additionally, a higher standard deviation of the on demand availability was measured. The high delay on the network results in few successful allocations as

displayed in Table 24. Compared to the first scenario, the second scenario achieves only 1.2% of the successful trades.

Experiment Id	Observation
1190403910890	CSA: 66, BSA_seller: 40, BSA_buyer: 40, RSA: 23 1951 observations
1190404122296	CSA: 14, BSA_seller: 16, BSA_buyer: 4, RSA: 4 24 observations
1190404999250	CSA: 66, BSA_seller: 51, BSA_buyer: 51, RSA: 58 1871 observations
1190405643250	CSA: 66, BSA_seller: 38, BSA_buyer: 37, RSA: 25 1109 observations

Table 24: Observation and involved agents for index computation

Surprisingly, the scenario with varying delay on the network links almost reaches the performance of the scenario with unlimited network bandwidth whereas the system performance decreases again in the scenario with constant high available bandwidth. The reason for this behavior is blocking policy during service and resource discovery. In scenario 3, the varying network bandwidth controls the number of proposals for the requesting agent. A high number of messages on a link come along with high message latency. This high latency delays broadcast messages to be delivered on network link which doesn't have enough capacity. The services are available to other requestors which have a better connection with lower latency to a given node.

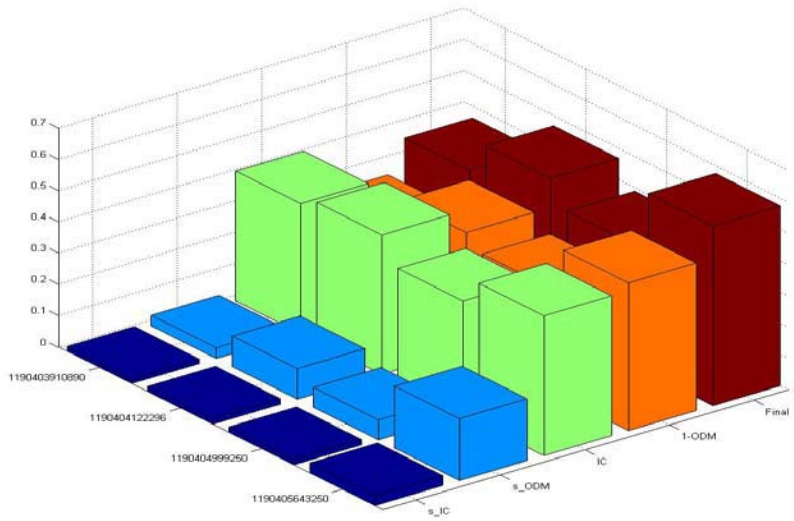

Figure 28: Final bar graph of 4 different bandwidth configurations

The influence of the blocking policy during discovery phase influences again the results of scenario 4. The same bandwidth on all nodes leads similar infrastructure costs like in the first scenario. The mean radar diagram of Figure 29 evidences better allocation times, allocation rate and provisioning time. But, higher deviations as shown in Figure 30 compensate these positive effects.

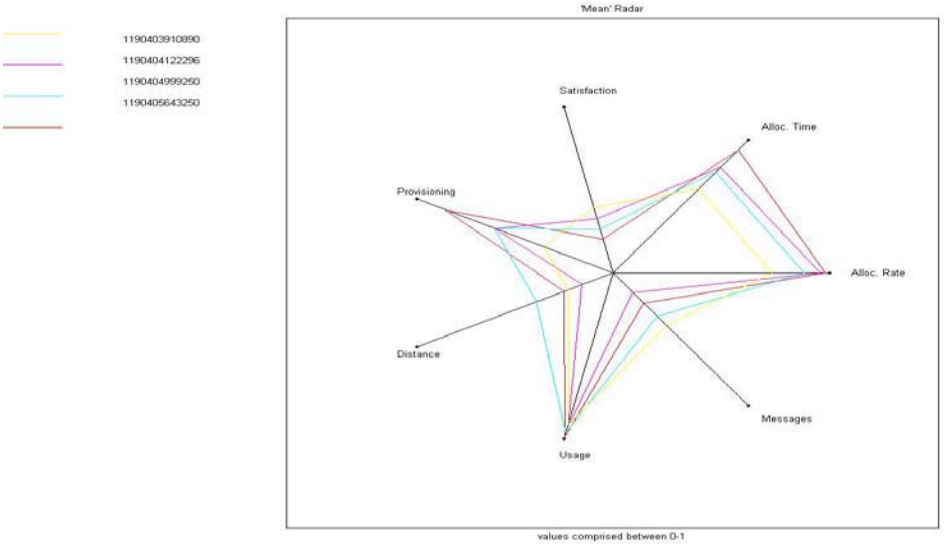

Figure 29: Radar plot of normalized mean values for seven selected metrics; four simulation runs with different bandwidth configurations are compared.

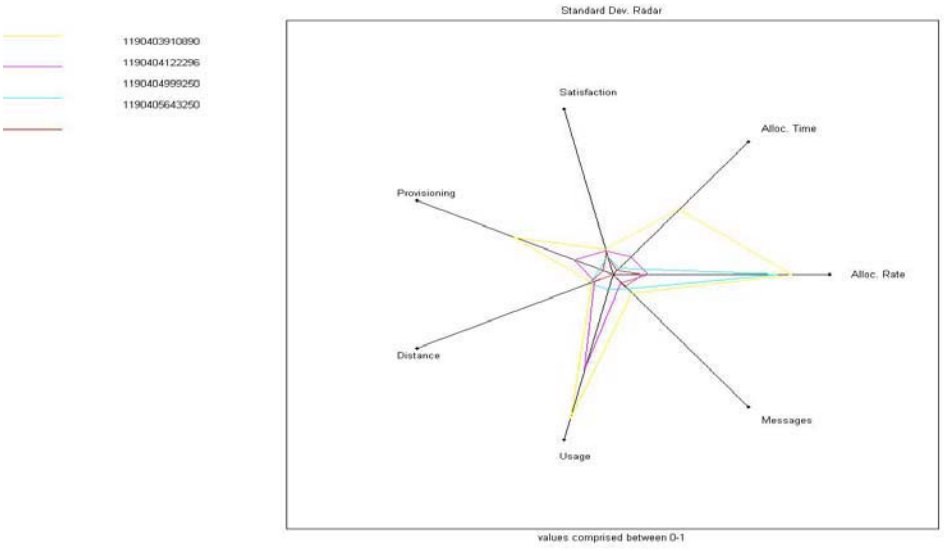

Figure 30: Radar plot of normalized standard deviation values for seven selected metrics; four simulation runs with different bandwidth configurations are compared.

In scenario 4, the number of observations is 43 % lower than in the best scenario. There is no clear evidence for this low number of observations yet because a similar number of unique agents trade. After a competition on the service market, the basic services compete for resources on the resource market. This could lead to a negotiation timeout on the resource market because the discovery timeouts are extended in case of no proposals in the inbox queue.

A clearer picture shows scenario 3. The varying delay on the network leads to a higher number of agents involved in trades with complex services and basic services. This scenario even outperforms the scenario 1 with unlimited bandwidth. More than the double numbers of resource agents are involved in trades. Therefore, scenario 3 shows the most distributed behavior. As expected, only a few agents are involved in the trades of scenario 2. There is a large gap between the two roles of the basic service. The limited bandwidth the basic service seller is able to sell his services, but the basic service buyer is not successful in making agreements on the resource market. Only one fourth of the successful service market trades reach an agreement on the resource market.

Summarizing up, varying bandwidth helps the catallactic strategy to achieve good results because it reduces the competition on the service and resource markets. Higher competition leads to lower performance on the investigated scenario.

3.6.8 Evaluation of the decentralized approach with different agent distributions

The following experiments evaluate the decentralized allocation approach for a set of agent distributions. Different distributions of complex services, basic services and resources services are evaluated concerning their influence on the final social utility index. The topology for all agent distributions has 400 nodes. The bandwidth of the network varies between 100 and 500. The message size was set to 2. This reduces the influence of the network to a minimum. The service and resource market configuration equals the configuration used the previous sections. Therefore, the configuration is not explained here again. The complex service dispatcher submits 10000 complex services requests in each simulation run. A total number of 300 agents are divided into 100 complex service instances, 100 basic service instances and 100 resource instances. The build-in distributions of the automated scenario generator are used to assign the agent instances to network nodes. As described in Deliverable D2.3, the automated scenario generator supports the following distributions:

- Uniform. The site for the agent is chosen using uniform probability distribution.
- Links (dir). The site for the agent is chosen with probability proportional to the number of site links. The more the site is connected, the greater the probability to hosts agents.

- Links (inv). The site for the agent is chosen with probability inverse proportional to the number of site links. The more the site is connected, the smaller is the probability to host agents.

- Dist (dir). The site for the agent is chosen with probability proportional to the distance between the site and a pivot site (the more the distance, the greater the probability).

- Dist (inv). The site for the agent is chosen with probability inverse proportional to the distance between the site and a pivot site (the less the distance, the greater the probability).

It is possible to assign a different distribution to each agent role. Table 25 gives on overview of the selected distributions. Experiment 2 uses uniform distribution for all agents which is the configuration used in all previous experiments. This experiment is intended to be the reference. Experiment 1 and 3 change the distribution for complex service and basic service agents whereas experiments 4 and 5 analyze the behavior of different resource distributions.

Experiment Id	Parameter setup
1190458974250	CSA: links (dir), BSA: distance (dir), RSA: uniform
1190464452609	CSA: uniform, BSA: uniform, RSA: uniform
1190469414062	CSA: distance (inv), BSA: links (inv), RSA: uniform
1190470099828	CSA: uniform, BSA: uniform, RSA: links (dir)
1190472995750	CSA: uniform, BSA: uniform, RSA: distance (dir)

Table 25: Mapping of experiment ids to parameter setup

Using this setup, two experiments are executed. The first set uses a broadcast hop limit of 4 and the second set a broadcast limit of 2. The hop limit of 2 will reduce the possibility of requestors to receive proposals and increase the influence of the agent distributions.

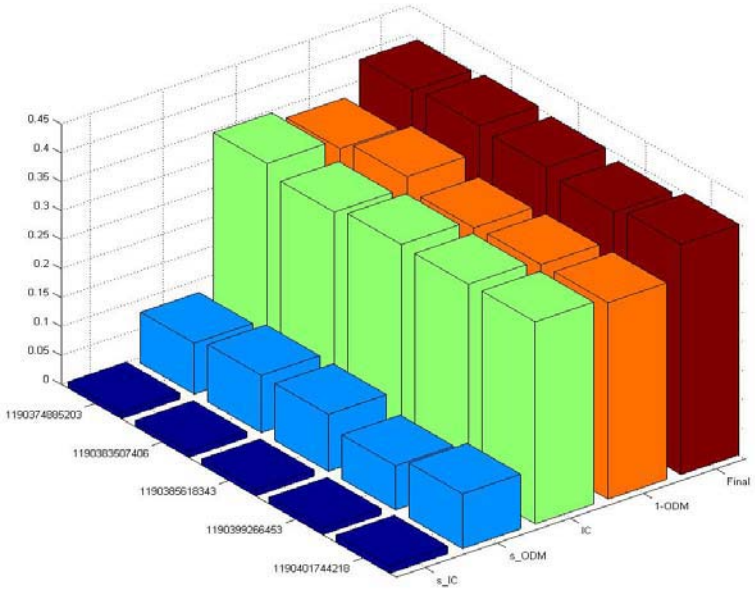

Figure 31: Final bar plot for 5 experiments with different agent distributions and 4 hops broadcast limit

Figure 31 presents the indexes for the different agent distributions and 4 hops broadcast limit. All experiments lie in between a small value range. Experiment 4 with more resources on better connected nodes achieves best performance. The reference experiment with all agents distributed uniformly shows worst performance.

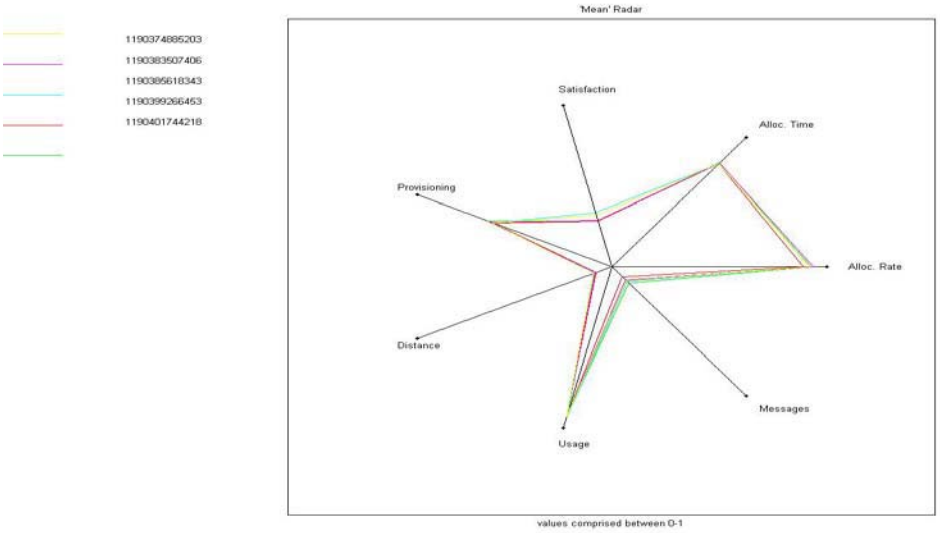

Figure 32: Radar plot of normalized mean values for 7 selected metrics; 5 simulation runs with different agent distributions and a hop count of 4 are compared.

Similar results show Figure 32. All values are close together, no experiment outperforms significantly another one. Deviations of the trading display Figure 33. The deviation of the allocation rate is less than then deviations in any other experiment. Most deviations are measured in experiment 2 with its uniformly distributed agents.

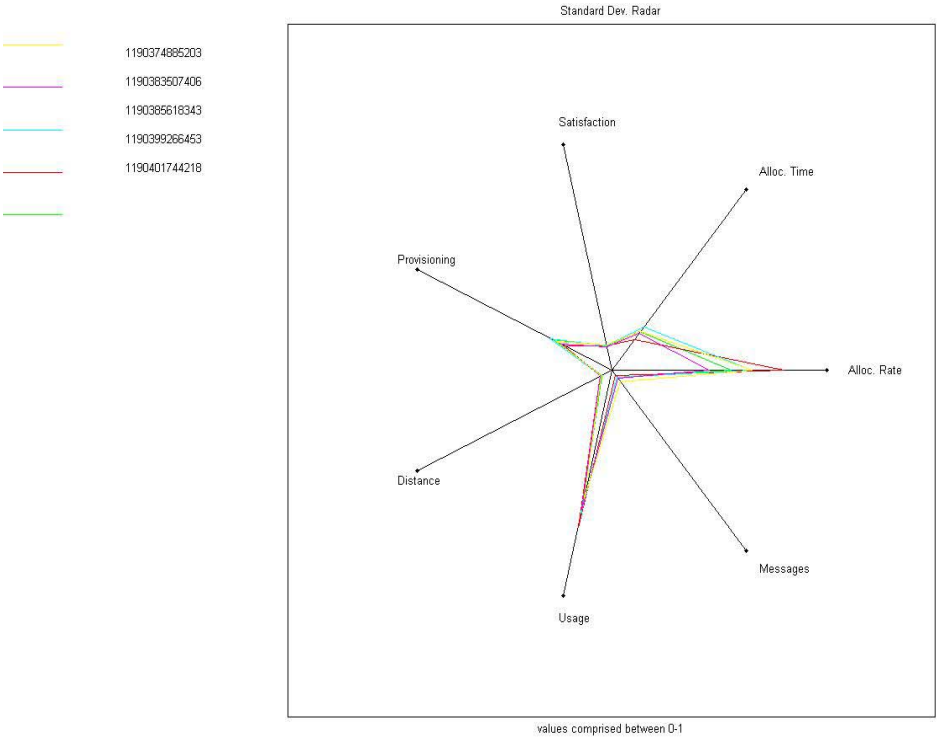

Figure 33: Radar plot of normalized standard deviation values for 7 selected metrics; 5 simulation runs with different agent distributions and a hop limit of 4 are compared.

The observation collected in Table 26 give also no clear picture of the agents distributions influence. The number of observations slightly varies between the worst value in the uniformly distributed agents experiment and the experiment 4. Also the unique number of trading agents is best in the experiment 4 by a high number of agents involved in trades at the same time.

Experiment Id	Observation
1190374885203	CSA: 96, BSA_seller: 79, BSA_buyer: 76, RSA: 69 16785 observations
1190383507406	CSA: 98, BSA_seller: 89, BSA_buyer: 76, RSA: 72 16398 observations
1190385618343	CSA: 98, BSA_seller: 73, BSA_buyer: 69, RSA: 65 16958 observations
1190399266453	CSA: 100, BSA_seller: 89, BSA_buyer: 85, RSA: 77 17951 observations
1190401744218	CSA: 99, BSA_seller: 81, BSA_buyer: 76, RSA: 70 17118 observations

Table 26: Observation and involved agents for index computation

In the experiment set with 4 hops broadcast limit, no clear evidence could found. Therefore, the number of hops was reduced to 2. All other parameters remain the same. Table 27 gives the overview of the experiments and their parameter setup.

Experiment Id	Parameter setup
1190458974250	CSA: links (dir), BSA: distance (dir), RSA: uniform
1190464452609	CSA: uniform, BSA: uniform, RSA: uniform
1190469414062	CSA: distance (inv), BSA: links (inv), RSA: uniform
1190470099828	CSA: uniform, BSA: uniform, RSA: links (dir)
1190472995750	CSA: uniform, BSA: uniform, RSA: distance (dir)

Table 27: Mapping of experiment ids to parameter setup

In general, the reduction of the hop limit increased the final social utility index by a small number. As indicated in the 4 hop scenario set, the gap of on demand availability between the worst and best case increases. Both, the resource distribution experiments 3 and 4 increase their on demand availability and decrease their non on demand availability respectively. Also the low deviation of the on demand availability is emphasized for experiment 4.

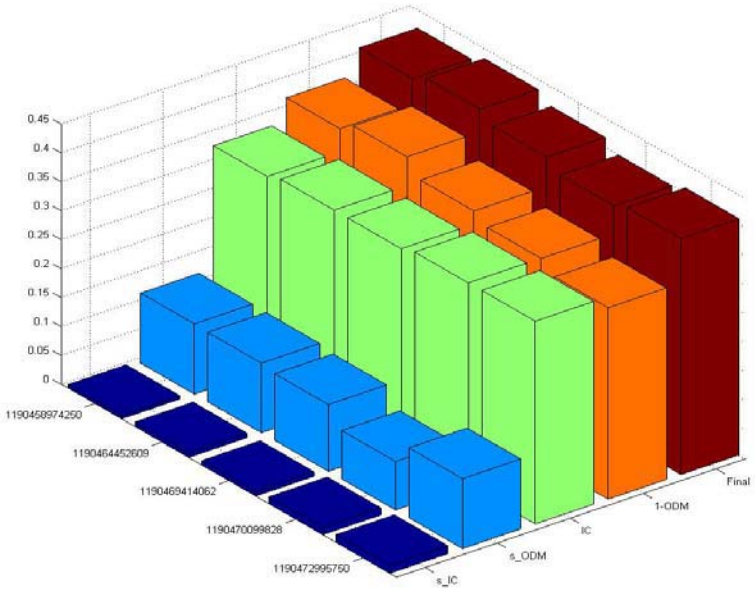

Figure 34: Final bar plot for 5 experiments with different agent distributions and 2 hops broadcast limit

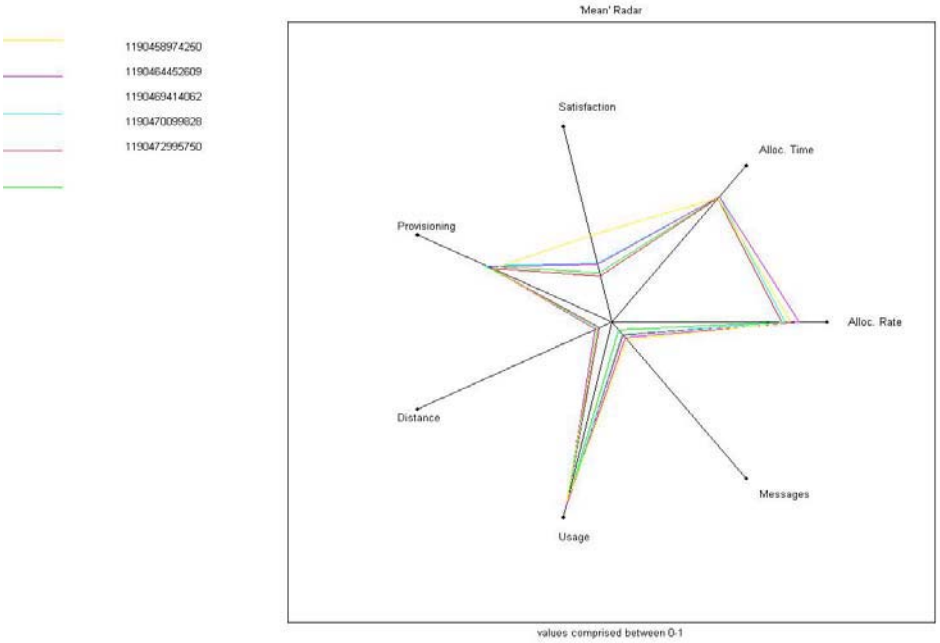

Figure 35: Radar plot of normalized mean values for seven selected metrics; 5 simulation runs with different agent distributions and a hop count of 2 are compared.

Figure 35 and Figure 36 depict the measured mean and standard deviation. The agents of experiment 1 achieve on average better satisfaction than the agent of all other experiments. The selected trading partners achieve good results mainly on the resource market. As already seen in the last experiment set, main deviation shows the allocation rate. The trading agents gain high allocation rates and low standard deviation in experiment 2. The larger distance and number of message deviation compensate the better allocation rate deviation.

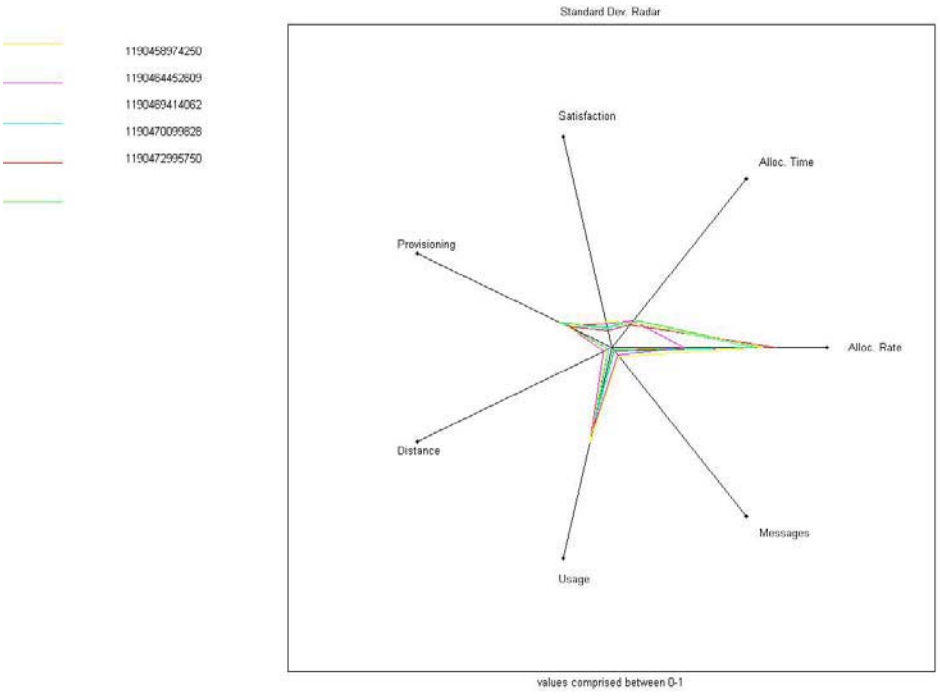

Figure 36: Radar plot of normalized standard deviation values for 7 selected metrics; 5 simulation runs with different agent distributions and a hop limit of 2 are compared.

Table 28 shows significant influence of the service and resource distributions on the number of agents involved in agreements and the total number of observations. The highest number of observations was measured in the experiment 4, which gives evidence for being a good strategy to place resources on good connected nodes. In experiment 1, complex services select only a subset of the available basic service agents, whose number decrease again on the resource market. Not every successful basic service seller can find a resource instance on the resource market. This gap between service and resource market is even worse in experiment 2 with its uniform distributions. A strong competition for resources lead to this result. The worst number of observations shows experiment 5. Choosing a pivot site right in the center of the network will place the resources at the network border. The basic service agents are not able to reach enough resources within the 2 hop broadcast limit.

Experiment Id	Observation
1190458974250	CSA: 81, BSA_seller: 59, BSA_buyer: 45, RSA: 45 12235 observations
1190464452609	CSA: 84, BSA_seller: 70, BSA_buyer: 44, RSA: 46 12561 observations
1190469414062	CSA: 80, BSA_seller: 60, BSA_buyer: 43, RSA: 44 11956 observations
1190470099828	CSA: 100, BSA_seller: 87, BSA_buyer: 85, RSA: 79 16621 observations
1190472995750	CSA: 77, BSA_seller: 59, BSA_buyer: 39, RSA: 42 11296 observations

Table 28: Observation and involved agents for index computation

In the evaluated scenario, no clear evidence of different agent distributions could be found. Possible drawbacks of an increasing distance to trading partners could be overcome with an increasing hop limit. Resources should be placed on good connected nodes which increases the number of successful allocations.

3.7 Market mechanism implemented in the prototype

The prototype has been evaluated with three different decentralized economic agent implementations (see Table 29).

Mechanism	**Description**
Catallactic	Catallactic agents, which maintain a complex strategy for negotiation, evolved trough evolutionary learning.
ZIP	ZIP agents, which employ a token based protocol to coordinate the issuing of bids/offers, which are then cleared upon the token completing each round. A previous implementation of ZIP-based agents worked with only local information but considered the resource usage in the price determination strategy.
CNet	Basic Contract-Net agents using a simple offer/demand protocol.

Table 29. Economic agents implemented in the CATNETS prototype.

In the following sections the three different agent types are outlined.

3.7.1 Contract-Net (CNet) simple offer/demand agents

The Contract-Net protocol (Figure 37) 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 conclusion of this period, the initiator selects from the set of collected proposals the best one, informing the winner. In 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.

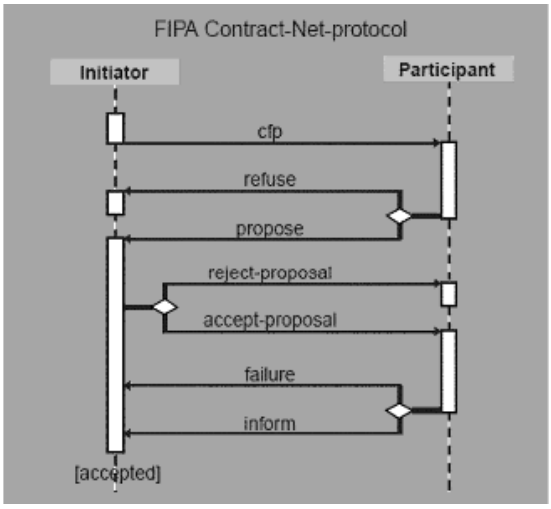

Figure 37. Contract-Net [taken from FIPA web site]

3.7.2 Zero intelligence plus (ZIP) agents

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 from seller Smin and the maximum price of the shouted bids from buyer Bmax. The bidding algorithm calculates the new price P(t+1) using the Smin and Bmax values. The algorithm implemented in the ZIP agents of the prototype is listed in Figure 38. A datailled description of the algorithm is outlined in deliverable D3.3.

```
Algorithm 1: 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 = • * priceChange + (1-•) * • * (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 = • * priceChange + (1-•) * • * (PT-P(t));
 P(t+1) = minimum (P(T)+priceChange, budget) ;
```

Figure 38. Bidding algorithm for BS (buyer) and CS (seller) implemented in ZIP agents.

3.7.3 Catallactic Agents

The catallactic agents and their implementation are described in the deliverbales D1.1 and D1.2. The reader is referred to those deliverables for more information about the catallactic allocation approach and its implementation.

3.8 Evaluation of the market mechanism in the prototype

We have carried out experiments with the prototype using the three different economic agent types. In the following sections, the experiments and results are explained.

3.8.1 Experiments with the Contract-Net simple offer/demand agents

The goal of the experiments is to show the performance of the GMM (Grid Market Middleware[5]) as an automated economic-aware resource management tool by means of the the DataMining Grid prototype application. We evaluate the ability of the Contract-Net based negotiation protocol for stabilizing fair prices in the Grid service, trading in different scenarios. 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 2.80GH and 2

[5] The middleware of the Catnets prototype is called Grid Market Middleware (GMM) in the context of that paper.

GB of memory. The server farm nodes are connected by an internal Ethernet network at 100Mps. The topology is a mesh: All nodes are interconnected. 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 4 nodes. Two nodes host a BS each and the Data Mining Web Service and other two nodes host the CSs, access points and clients. The Web Services are exposed in Tomcat servers. The experiments consist in launching 2 clients concurrently, which use each one of the CS as broker. Each client makes 100 requests to the CS in intervals of 2 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.

We have two different scenarios in the dedicated resource model. The demand indicates the rate at which new CFPs are issued. If the proportion demand rate/Data Mining Web Service execution time is lower than 1, then the resources are potentially able to handle all the demands. In the contrary case, the demand exceed the offer of resources, hence several CFPs will be disregarded even in the case they meet the pricing criteria of the BS.

In a first experiment, we set up a demand rate lower than 1. Resources therefore are able to cope comfortably with the demand. We set up two CSs with initial bidding prices of 75, and 2 different BSs, with initial offer prices of 60 and 90 respectively. In Time (sec**)** Figure 39, it can be seen that the price quickly stabilizes to a “fair” value around 72. This result holds true independently of the initial prices on the CSs. As for the initial prices of the BSs, the BS1 starting with a lower price trades more in the beginning, but trying to increment its surplus it soon reaches the equilibrium price. As for the BS2 starting with a higher price, it does not trade in the first CFPs which leads to a continuous price dropping towards the equilibrium price. After stabilization, the CSs get their CFPs for resources granted, provided the bid equals at least the offered price, which is true during most of the experiment.

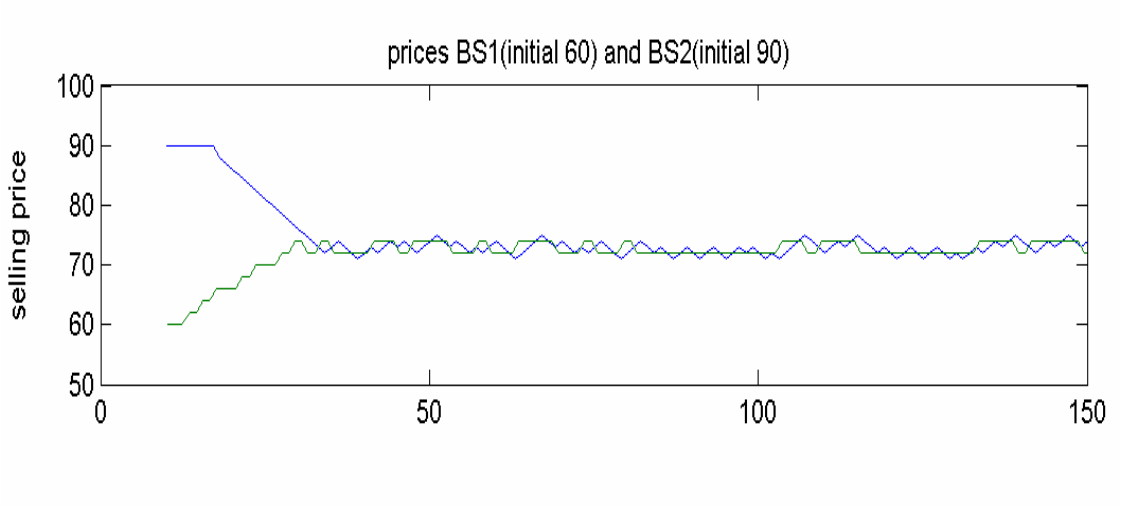

Figure 39. Evolution of prices vs time for a low demand rate.

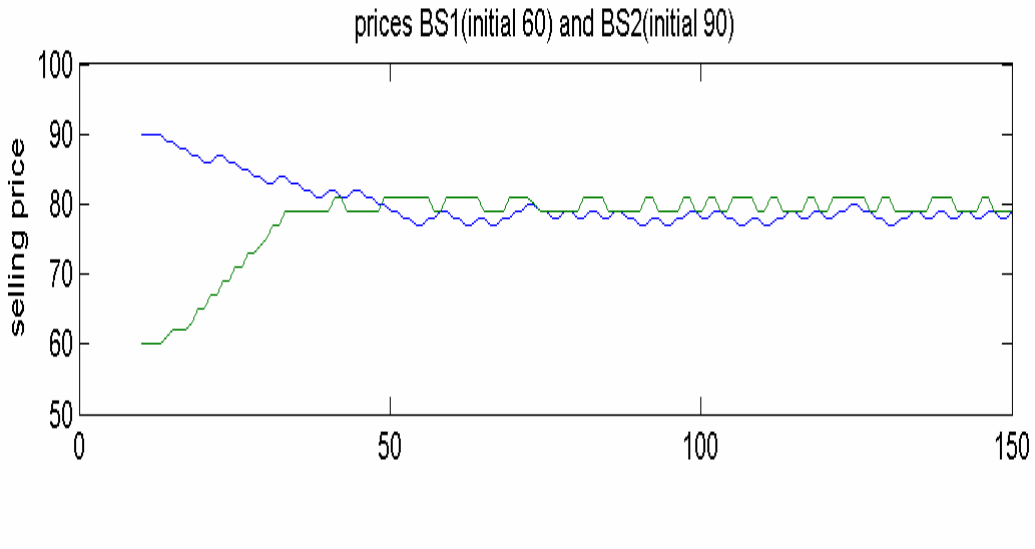

Figure 40. Evolution of prices vs time for a high demand rate.

In a second experiment (Time **(sec)**
Figure 40), we set up a high demand. In this case the resources of the providers are not enough to completely meet the demand. This does however not make the prices increasing indefinitely, since the successful trades make the CSs react trying to decrease bids. We set up agents with the same initial prices as in the previous example. The price stabilizes also quickly, but in this case to a higher price at around 80, due to the resources scarcity (Time **(sec)**

Figure 39, right diagram). This result holds true independently of the initial prices on the CSs. In this scenario, the CSs gets just half of their CFPs for resources granted, but this is evenly distributed between the two CSs, which is a fair resource share.

The results of both experiments demonstrate how a simple decentralized economic algorithm can be plugged into the GMM infrastructure in order to allocate resources to client in service oriented applications, by achieving automatic and fair trading of resources between Grid clients and Grid service providers, mediated by the CS and BS agents, respectively.

3.8.2 Experiments with the ZIP agents

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 2.80GH 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. 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 8 nodes. Four nodes host a BS each and the Data Mining Web Service and other four nodes host the CSs, access points and clients. The Web Services are exposed in Tomcat servers. The experiments consist in launching 4 clients concurrently, which use each one of the CS as broker. Each client makes requests to the CS and leaves the market after a successful trade. It will re-enter a proceeding round with the probability of 1/3. 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.

The goal of the experiments is to show the performance of the GMM as an automated economic-aware resource management tool by means of the Data Mining Grid prototype application. The extended ZIP agents are expected to show an effective and fair trading, which can be measured with the price and the allocation rate of each agent. Varying the technical parameters of the environment, we expect price adaptation of the agents in the marketplace.

3.8.2.1 Idealized experiments with idle resources

The experiments are sensitive to a competitive use of other processes, because this might cause an increase of the Data Mining WS execution times. Therefore we make first experiments with idle resource, which guarantees the stability of Data-Mining Services execution times (Figure 41, Figure 42). We see how high load in the Data Mining Services redues the resource availability, hence rising prices.

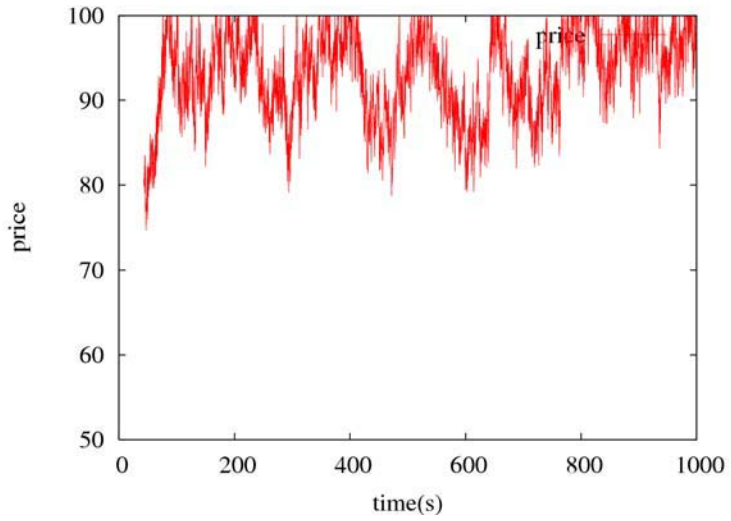

Figure 41. Price evolution with varying offer with constant demand rate ½ - resource execution time of 3000ms.

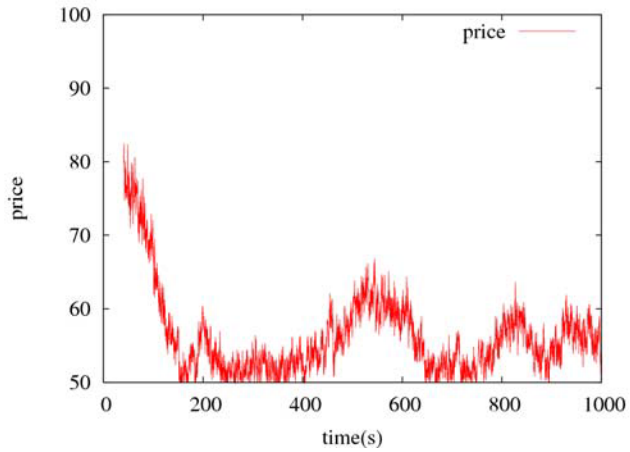

Figure 42. Price evolution with varying offer with constant demand rate ½ - resource execution time of 100ms.

Besides the effect of changing the offer, also the variation of the demand for the resources needs to be proved. Therefore we change the probability that a CS re-enter the market (by issuing a new demand) after a successful trade. In Figure 43 the demand rate

probability of re-enter the market is 1/6, which keeps the amount of the CS low and decreases the price. Figure 44 shows the price increasing when the CS re-enter the market after every successful trade (probability of 1/1).

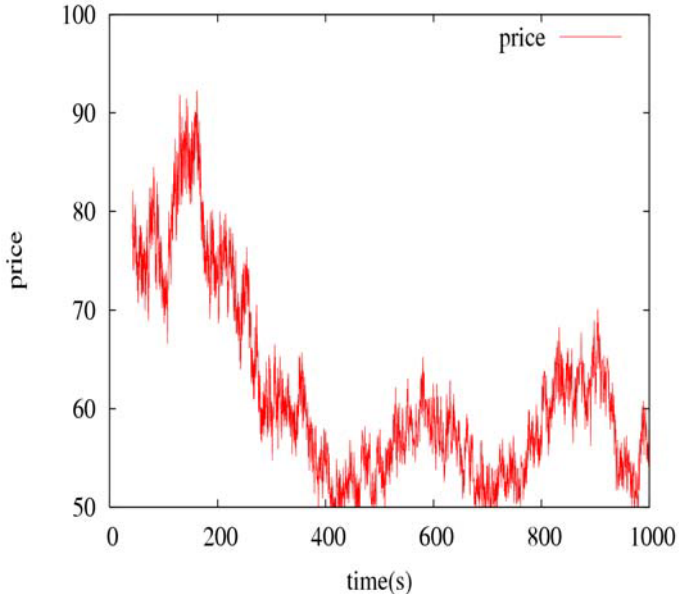

Figure 43. Price evolution with varying demand rate with constant executionTime 1000 ms – demand rate 1/6.

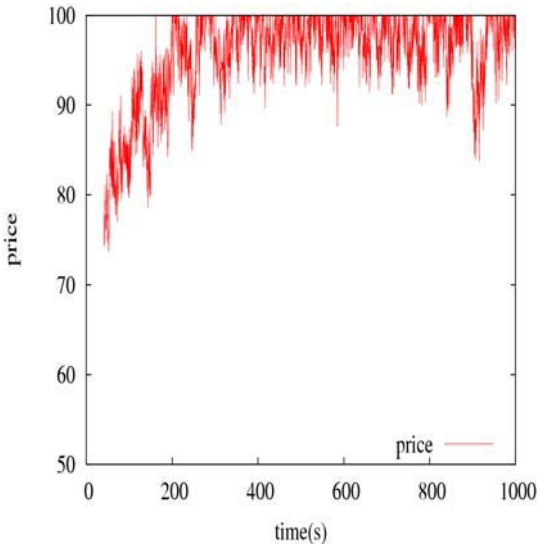

Figure 44. Price evolution with varying demand rate with constant executionTime 1000 ms – demand rate of 1.

3.8.2.2 Adaptation to different constrains

In this section, the experiments illustrate the adaptation of the prototype for a changing environment. Here, the execution time of the Data-Mining Services varies, mapping to real scenarios where input data-sets to be processed might differ in size. Simulating such cases, the execution time of the resources will vary during the running time of the experiment. Every 200 seconds it changes iteratively the executions time from high (3000ms) to very low (100ms).

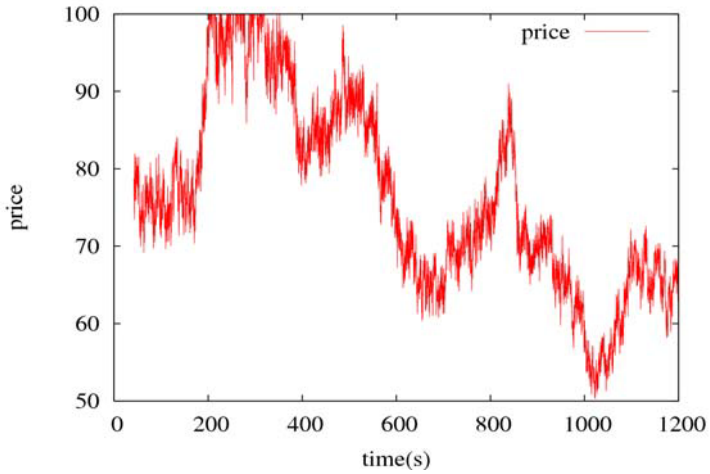

Figure 45. Varying task load (WS execution times) dynamically. t = 0 – 450 (phase 0): stabilization t = 450 – 650 (phase 1): WSexecTime: 100; t = 650 – 850 (phase 2): WSexecTime: 3000; t = 850 -1050 (phase 3): WSexecTime: 100; t = 1050 -1200 (phase 4): WSexecTime: 3000.

In Figure 45, it can be observed that after stabilization phase of about 450 seconds (phase 0), showing price adaptation to varying market constrains in form of task loads (the WS execution times). If the execution time of the resources is short (like 100 milliseconds), then the market contains many offers. Consequently the prices of the product decreases. Contrarily, decrementing the supply by setting the execution time to 3000 milliseconds leads to an increasing price.

3.8.2.3 Process competition

Increasing the realism of the environment, we consider an experiment were the nodes in the cluster run other competing processes which influence resource performance. This has an impact on resource offer which should be considered by the agents. We show how agents effectively react to the process competition by adapting prices (Figure 46 and Figure 47). In this case, there is not a clear pattern of price evolution, but generally we see that process competition reduces the number of matches (and resulting price risings), as expected in a more realistic environment.

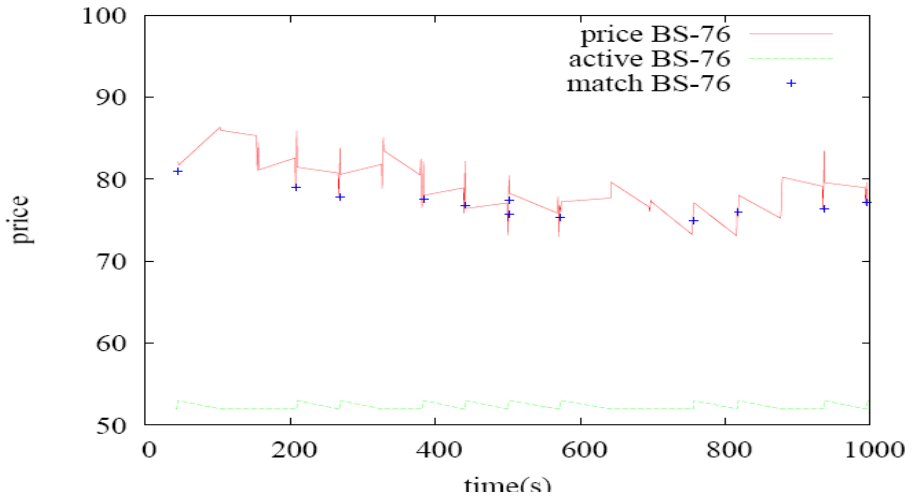

Figure 46. BSs prices with competing process.

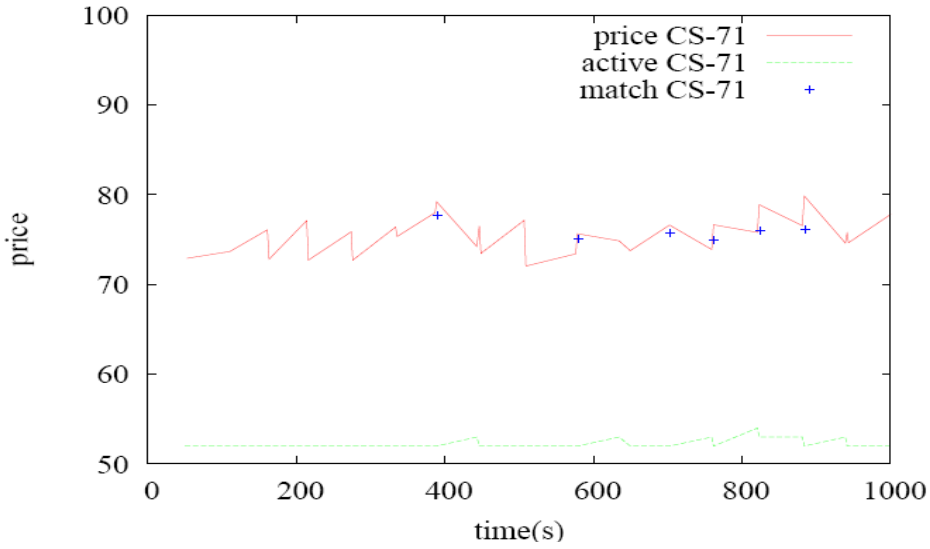

Figure 47. BSs prices with competing process.

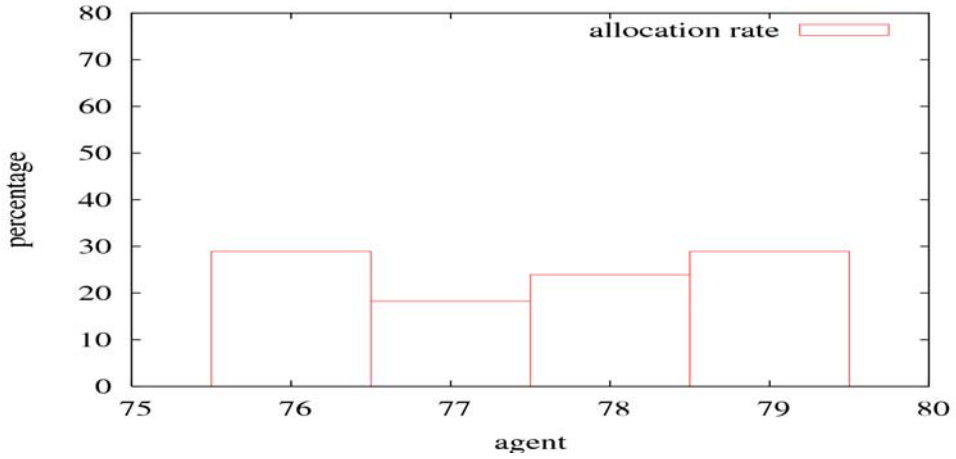

Figure 48. Allocation rates in competing process experiment. Allocation rate of CS.

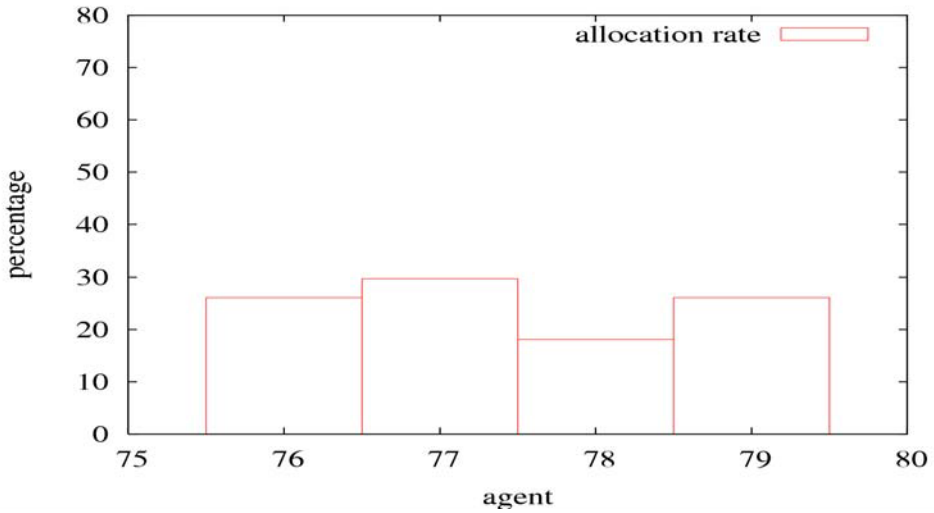

Figure 49. Allocation rates in competing process experiment. Allocation rate of BS.

The allocation rate shows the distribution of over 4000 matched trades. A nearly equal distribution of the resources to the CS (Figure 48) can be seen as well as the nearly equal distribution of the bought resources from the BS (Figure 49) can be seen. Even in a real application under some real process competition, an almost fair allocation is obtained. Figure 48 shows how CSs share almost evenly the trades, and the same pattern applies for BSs in Figure 49, which means the algorithm is not isolating agents from the market and enables the participation of all the agents in trades.

3.8.2.4 Evaluation of ZIP agents

The results of the three experiments demonstrate how a simple decentralized economic algorithm based on ZIP can be plugged into the GMM infrastructure in order to allocate resources to client in service oriented applications, by achieving automatic and fair trading of resources between Grid clients and Grid service providers, mediated by the CS and BS agents, respectively.

Furthermore, the results show that the agents react to changes in the economical environment. The accepted price reflects the variations in demand (trough demand rate) and offer (trough varying execution time of the services, which results in varying resource availability). It can be seen that the price increase when the demand increases (Figure 44) and that the price also increase when offer decreases (Figure 41), as a result of the services consuming more time. Nevertheless, the distribution of allocations between buyers and sellers remains proportioned (Figure 48 and Figure 49), as it should be in a fair market. It follows that the prices will increase in case of large-scale failures or delays. Moreover, this automatic price correction behavior is able to react to dynamic varying conditions in the underlying Grid resources (Figure 45).

3.8.2.5 Evaluation of reource-usage dependent ZIP-like strategy

In this section, we report on early experiments made with the prototype using a resource usage dependent price determination strategy. The agents were later replaced by other agent implementation working with a dedicated resource-model, i.e. the resource was either available for sale or completedly locked by a service execution. Reasons for this choice were to work both in simulator and prototype with the same model, and secondly, from the experiments made it appeared to be difficult to control the experimental results due to the feedback introduced from real resource usage (partially by other users working on these non-dedicated machines) in the price computation.

In this resource usage depented price strategy, the agents, besides taking into account the success of previous sales, read the current CPU usage with was taken into account for computing the price for negotiations.

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 influenced by the node's utilization. Then, the pricing model is combined with the demand. If a service agent sells 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.

We have deployed the Grid Market Middleware in a Linux server farm. Each server has 2 Xeon processors and 2GB of memory. The machines in the server farm are connected by an internal Ethernet network at 100Mbps. Three basic services (BS) are deployed on

three servers (BS-74 on node 74, BS-75 on node 75, BS-79 on node 79, respectively), and two complex services (CS) are launched on two other servers (CS-72 on node 72, CS-73 on node 73, respectively). On each machine with a BS we also deploy a web service representing the application, which performs a CPU intensive calculation. These web services are exposed in a Tomcat server. Access to execute these web services is what is negotiated between complex services (buyer) and basic services (seller).

We run an artificial background load on two of the nodes (node 79, node 75) configured for 50% and 100% CPU usage to simulate background activity. This is chosen since in such a setting the behaviour of the agents should lead to load balancing of the web service executions.

The experiments consist in launching 2 clients (represented by complex services CS-72 and CS-73) concurrently as clients. Each client performs 50 requests in intervals of 15 seconds. Whenever a client wins a bid with a service, it invokes the web service in the selected node. The data obtained from the experiment with the performance measuring infrastructure has been the following:

1. *allocation*: an entry by each successful negotiation with a basic service, reported by the complex service
2. *price*: a periodic report of the price of the basic services
3. *utilization*: a periodic report of the CPU utilization given by the resource agents
4. *execution.time*: time needed to actually execute the service, reported by the complex service (transaction-based).

The experimental results are illustrated in the following figures. Figure 50 shows the load (% CPU usage) on the three nodes (74, 75, 76). A background load of 50% and 100% in nodes 79 and 75, respectively, can be observed. The up-going spikes which can be seen in the load of node 79 and node 74 correspond to the execution of the negotiated web services on these nodes.

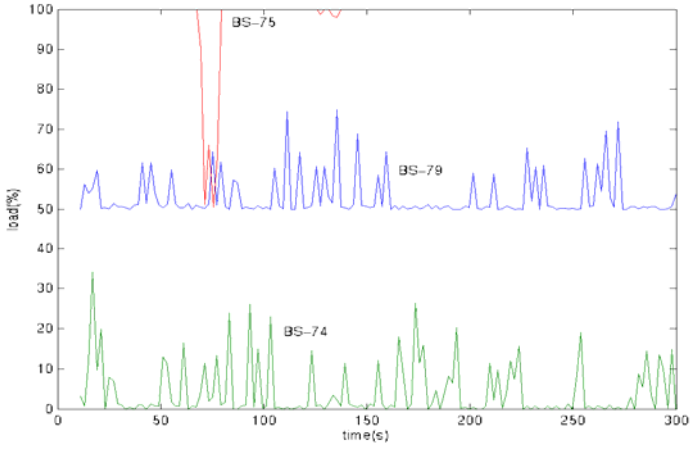

Figure 50. Load on nodes 74, 75, and 79. Node 79 and node 75 are with 50% and 100% background load, respectively.

Figure 51 shows a zoom on the price of the basic services. It can be observed that the price calculation of the agents takes into account the success of past negotiations, where the price rise is made after a successful sale. The configured buyer price is 100 money units.

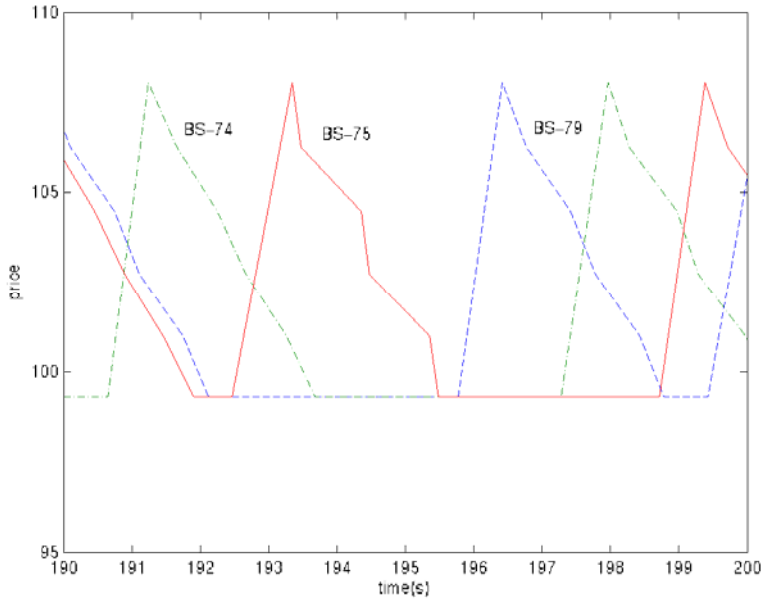

Figure 51. **Zoom on the price evolution of the basic services in nodes 74, 75, and 79.**

Figure 52 is to assess the expected load balancing behavior, which we should obtain with this setting. It can be seen that effectively the BS-74, which runs on the least loaded node, makes most of the sales. And the BS-75, which runs on the node with the highest background load, makes less sells than the other two basic services. We can see that the performance measuring framework achieves one of our goals which was revealing such behavior.

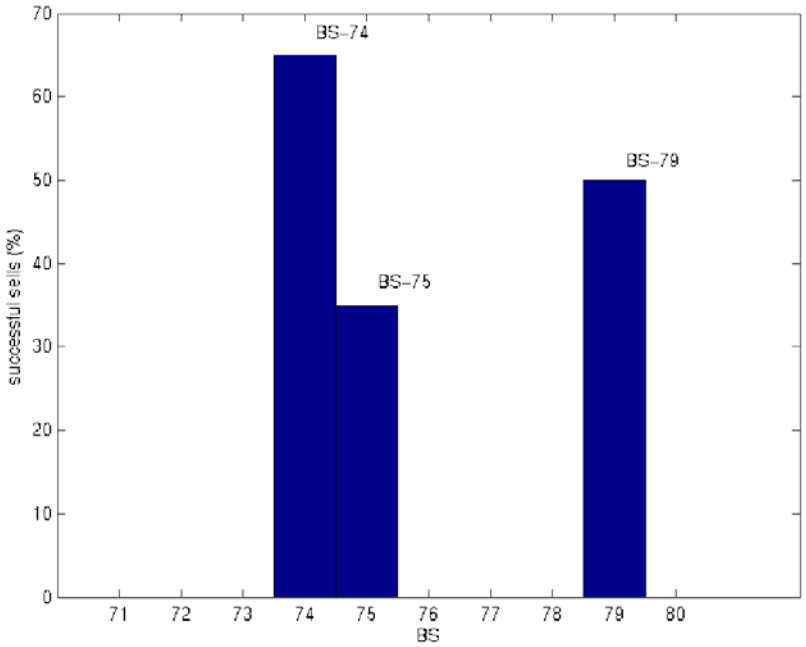

Figure 52. Percentage of sales of the three basic services. BS-74 which resides on the least loaded node, makes most of the sells.

Finally, we observe two metrics together in Figure 53: the successful sales by BS-74 and the execution of the sold service when invocated by the clients (the web service is executed on the same node 74). Successful sales by the BS-74 are indicted with a star symbol and are normalized here to the value 30 for easier visualization. An execution of the web service follows each successful sale and last approximately 4 seconds.

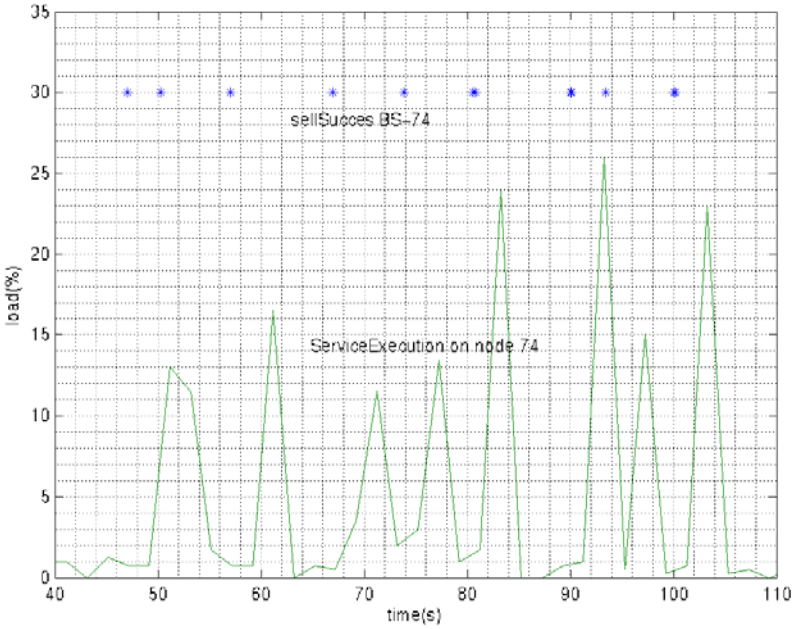

Figure 53. Successful sells of BS-74 and web service execution on node 74.

The experimental results demonstrate the two contributions we want to emphasize: First, the measurement framework achieves obtaining multi-level performance data of the distributed application. The data shown in the experiment is taken from the three main levels of the architecture: The time of the web service execution is measured at the application level. The evolution of prices is taken at the economics algorithm layer of the middleware. Finally, the load at each node is taken from the base platform layer. Secondly, the obtained data is useful for the analysis of the middleware and application (the setting was deliberately chosen to force load balancing behavior). We have seen that with the obtained data the expected behavior can effectively be observed.

3.8.3 Experiments with the Catallactic agents

The allocation rate measures for the catallactic agents the number of client request which have been completely handled by the GMM. This comprises first reception of the Request by the ApplicationProxy and its forwarding to a CS; then, the CFP issuing by the CS and the BSs answering collection /selection in the first market; finally, the complete negotiation CS-BS to reach a final agreement which fires the trade completion metric.

In order to calculate the allocation rate for the service market from the prototype experimental data, we count all the negotiation_end events containing a CS. This means that at least a Complex Service took part in the negotiation, implying that the trade was

on the first market. You need to take into account both the negotiation_end metrics fired by CSs or BSs. This happens since both CSs and BSs can close the negotiation.

We consider experiments with 1 CS, CS1, and two competing BSs: BS1 and BS2. In each scenario the client application issued 100 requests to the market. The application creates an applicationProxyAgent for each demand request to the CATNETS market. After each creation the application waits for the agent to finish, e. g. for the overall market request to finalize. The timeout for the market to answer is set to 4000ms. If the applicationProxyAgent did not finish the negotiation is viewed as a failure.The execution time determining the time a resource is blocked was 2000ms.

Each CS is able to sell one type of service, e. g. a generic service called cs1, whereas each cs1 service requires an instance of basic service bs1. The basic services are able to sell services of the type bs1 and require resources of type CPU for their sold basic service on the service market. This scenario is chosen due to simplicity reasons. The implemented agents and middleware infrastructure is however capable of trading more complex services as well.

Each agent involved in the CATNETS market was set to have an acceptable price interval of 25 – 40. The discovery timeout limiting the time an agent waits for the other market participants to answer a cfp was set to 5000ms.

The strategy of the agent was set as follows:
aquisitiveness: 0,5
satisfaction: 0,99
priceStep: 0,3
priceNext: 0,05
weightMemory: 0,9

The dynamic in agent evolution is expressed as a probability of mutation (set to 0,05) and of crossover (set to 0,2).

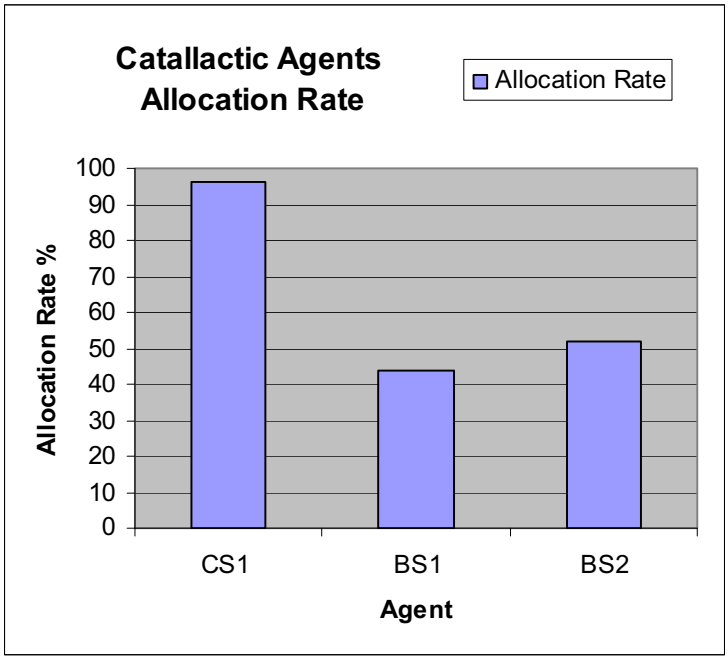

Figure 54. Experiment 1. Allocation rates.

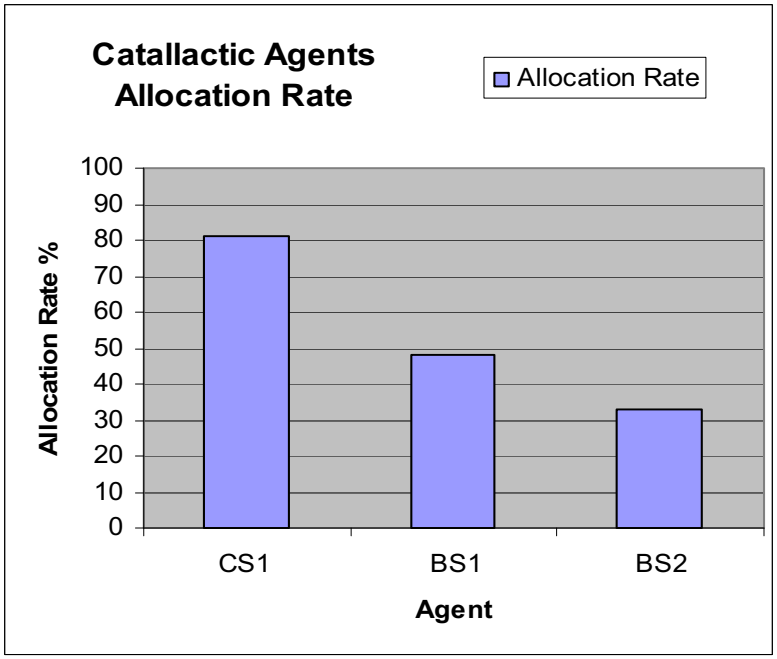

Figure 55. Experiment 2. Allocation rates.

As we can see in Figure 54, in the first experiment a very high percent of the requests by the CS got fulfilled, about 96% of success. As for the trade partners, the BSs shared more or less evenly the 96 trades fired. In the second experiment, Figure 55, we see 82% of success for the CS1, in this case the BS1 got slightly more allocation than BS2.

3.8.4 Comparison of Catallactic agent with ZIP agent

The two mechanism differ substantially, since the ZIP agents are exchanging a token between them in each round, and failure to fire trades in this round are counted as a request failure by the agent. This is quite different from the allocation in Catallactic agents, where each request by the CS conveys an iterative negotiation with the selected BSs until an agreement/or disagreement is reached.

However we can still compare how the succesfull trades are shared between BSs.For the ZIP agents, the allocation rate of Figure 48 and Figure 49 shows the distribution of over 4000 matched trades. A nearly equal distribution of the resources to the CS can be seen as well as the nearly equal distribution of the bought resources from the BS can be seen. Even in a real application under some real process competition an almost fair allocation is obtained. If we compare this with the Catallactic agents, we see that the allocation rate between BSs is shared more or less evenly in both cases between the two BSs (Figure 54 and Figure 55). However notice that both protocols are very different, since each round ZIP agents only perform one bidding, while each round in the Catallactic agents encompasses a full negotiation between the CS and the best BS answering the CFP.

A high level comparison between two ZIP agents and Catallactic/C-Net protocols reveals that ZIP agents encompases few number of messages (just an initial multicast and then the token exchange), while the C-Net agents require full cycles of CPF followed by answers to that CFP from all potential sellers, as well as the answer from the buyer selecting/discarding candidates. The Catallactic agents further add on top of C-Net protocol number of messages the successive bilateral bargainings required to reach an agreement between buyers and sellers. The Catallactic agents ensure in most of the cases the reaching of agreements after a number of sequential negotiations between a buyer and potential candidates, while ZIP agents and C-Net normally reach fewer allocation rates, but also generating less overhead.

4 Prototype evaluation

The assessment is on the following aspects: Implementation feasibility and available standards, and secondly performance.

4.1 Evaluation of prototype development

4.1.1 Architecture

The prototype has been build based on a layered architecture (see deliverables year 2 and year 3 of WP3). This architecture proposing a so-called P2PAgent layer for the technical services, an EconomicsStrategy Layer for the economic agents, and an EconomicsFramework layer for separation of both layers has allowed developing the prototype.

The benefits of defining this architecture have been the following:

- Following this architecture, the different agent types could be implemented by substituting mainly only the components referring to the EconomicsStrategy layer.
- The communication between the application and the middleware has been reduced to the communication to a Catallactic access point, which by means of the WS-Agreement specification transmits all required information to the middleware. The protocol describing the order of messages exchanged, however, is particular to the prototype.

The architecture has been described in the following papers: [ACC05] and [CCF05].

4.1.2 Catallactic-enabled applications

The catallactic paradigm as introduced in predecesor Catnet project has been implemented in a concrete middleware and prototype applications in the scope of CATNETS project. The experience gained on such endeavor has resulted in both deceiving and promising conclusions, shared in equal proportions.

In the deceiving side, important trouble during prototype calibration has come from the requirement of dealing with decentralized decision makers (the trading agents) in a real, networked, infrastructure. Such a development is pioneering, since state of the art approaches to fully decentralized markets based on bargaining agents have been based purely in simulations [DUA04],[PT98]. In that sense, “touching the ground” of real deployment has proven hard due to the uncertainty and lack of control in such

“engeneering with complexity” tasks. Emergent engeneering has been largely coveted by large distributed systems engineers [ECE05], but to date just initial steps in form of proposals [TU05], [WH05] have been achieved. In our view, and summarizing our experience, more advances trough extensive testing in both software lifecycle management and practical deployment tips need to be realized in orther to reach maturity.

In the promising side, we have been indeed able to design, implement and deploy a fully decentralized prototype incorporating emergence and self-organization using state of the art tooling. The GMM implementation has been proven as useful in several applications. The use of the Catallactic middleware has been shown by two applications: COVITE being available as Grid Service and Data Mining tools given as Web Services. Both of these applications have been Catallactic-enabled within the project. Our results are documented in more detail in the following papers: [JRC+05], [JRC+06], [JRC+07]. From the experience obtained, we have found that applications provided as services can be enhanced with reasonable effort to interact with the GMM. Additionally, the GMM is being useful as an infrastructure root for further development in Grid Markets research project, as in the case of SORMA [SOR07]. The open issue with the catallactic-enabled applications is to achieve improved system control, in the form of more predictable outcomes out of the emergent properties of the markets.

From the application point of view, the fact of having participants offer and request for application services and computing resources of different complexity and value in a distributed environment leads to the creation of interdependent markets. In such interrelated markets, allocating resources and services on one market inevitably influences the outcome on the other markets. A common approach of many other Grid market concepts is to allocate resources and services by relying on the presence of centralized resource/service brokers. However, the complex reality could turn such approaches useless, as the underlying problem is computationally demanding and the number of participants in a worldwide distributed environment can be huge.

Different examples of application scenarios can be constructed which benefit from using the Catallactic markets in combination with different auction mechanisms in the Grid. This leads to an advantageous flexibility in terms of fulfilling the requirements and needs of services and resources within the applications and hides all the complexity to the users. Let us consider an application scenario that requires a highly specialized service such as medical simulation service or visualization service, while another application requires a specific mathematical service. The mathematical service is more or less standardized and there are several suppliers offering this service, and an instance of a catallactic market could be initiated and based, for example, on a normal double auction. The medical simulation service, however, does not have many service suppliers; therefore the liquidity of the market trading such services may be low. In such cases, an instance of a market could be initiated and be based on English auction mechanism. Other types of applications enable creation of Virtual Organizations (VOs) for planning, scheduling, and coordination phases within specific projects or businesses, and allows the users of a VO to interact among them for the duration of VO. The ability of a free-market economy to adjudicate and satisfy the needs of VOs, in terms of services and resources, represent an

important feature that markets, through the auction mechanisms, can provide to. Such VOs could require large amount of resources which can be obtained from computing systems connected over simple communication infrastructure such as Internet. There could also be possibilities for these VOs to try maximizing their own utilities on the market.

In conclusion, catallactic-enabled applications are well motivated and address real need of current realistic Grid scenarios. We have developed the first prototype available which is able to deploy the complex catallactic behavior in real Grid applications. The experience gained is valuable as it is, but it can be also profited by engineers in the field of "engineering with complexity and emergence", where prototype implementations and deployments in real tesbed applications are increasingly necessary to advance the state of the art.

4.1.3 Standards

Service Level Agreements (SLAs) provide a contract between an application user requiring services/resources, and application providers determining what should be made available for external use. To enable service/resource sharing/usage in application environments, SLAs may be used to define: (a) requirements that such an application would place on services (and resources) owned by a third party; (b) check whether these requirements have been met during use. An SLA may also specify the penalty that a service provider may incur if terms in the SLA are violated.

Currently, SLAs are defined in a static manner, i.e. the terms within an SLA must adhere to strict constraints, and are monitored during application execution – such as in WS-Agreement. However, within many applications, it is often difficult to define such constraints very precisely, thereby leading to a large number of violations. There is a need to modify an agreement that had already been established, especially if the agreement is used at a time much later than when the agreement had been defined. These requirements relate to comparing the cost of re-establishing a new agreement vs. being able to adapt an agreement that is already in place. Secondly, there is a need to support flexibility in the agreement if an agreement initiator is not fully aware of the operating environment when the agreement is defined. In this case, the agreement initiator may not have enough information to determine what to ask for from a provider. This is likely to be the case when an agreement initiator or provider operates with imprecise knowledge about the other party involved in the agreement.

Specifications which have been applied for the development of the prototype have mainly used the concept of SLA using the WS-Agreement protocol. The specification allowed describing the services needed by the users' application. The protocol for the exchange of WS-Agreement messages between the application and middleware needs to be developed for further negotiation interaction, which has been identified as a limitation. A use of WS-Agreement in the Catnets prototype is reported in [JRC+07].

4.1.4 Implementation

The implementation of the prototype took advantage of the functionalities already provided by available toolkits, like Diet agents, JXTA, GT4 (the middleware selection process as well as the GMMs' early design with them is described in the first year deliverable of WP3 [Del05a]). When running initial experiments with the developed prototype, however, limitations in the practical use of these toolkits have been observed, like a limited number of messages which could be sent with Diet, and the difficulty of JXTA to work correctly with a small number of nodes.

The observed limitations of the Diet toolkit affected the initial design of the performance measuring framework. Another design has been finally implemented which did not rely on the messaging mechanism of Diet, such that in the current prototype we do not have a limitation concerning this issue.

The limitation of JXTA affects the scope of deployment of the prototype in the sense that in small scale scenarios the delivery of messages by JXTA is not reliable. In the environment of the cluster where the prototype has been used, the identified limitation did not appear by making use of particular JXTA messaging services for this context. From the experience obtained, for complete decentralized scenarios, DHT implementations like Pastry could have been a better choice for the implementation of the communication and search.

4.2 Evaluation of prototype performance

The prototype has been deployed in a cluster of Linux machines. Several experiments have been made with different type of economic agents, and varying the different parameters of the experiment configuration. The results of the experiments with the Catallactic agents and other agent types are covered in section 3.8.

The experiments showed the behaviour of the measured parameters for concrete experimental settings. The character of the experimental results is rather that of confirming the implementation feasibility in terms of a prototype. The comparison of different agent approaches by means of the developped prototype is difficult. A large number of parameters remain uncontrolled due to the use of a real environment. The difficulty in configuring the Catallactic agents in the sense of being able to chose the most appropriate values for the parameters makes it appearing too early for stable quantitative studies.

The level of the prototype imposed certain constraints on the system compared with what could be a production-quality implementation. As a consequence, quantitative results are only achieved within the scope of the prototype. Our performance results are experiments taken at different stages of the prototype development. Although they revealed the behavior of the system in the current experimental configuration, they gave feedback on the implementation and hints on the complexity of applying Catallaxy in real systems.

We have developed several scripts for deploying the GMM. These scripts were toroughly described in D3.2. Over the last year we have tailored the scripts as a tool for deploying the three differente flavours of decentralized economic agents in LANs and cluster. These scipts allow for the full deployment, remote experiment execution and metric collection of the GMM and application WSs. They have been released with the CATNETS software with a companion documentation and tutorial (see WP3 year 4 Annexes).

The flow of a typical prototype experiment is quite simple, if we consider the large number of automated steps from the original client request till the moment when the EPR for service execution is returned back. An example of an expriment with 4 nodes is the following: Two nodes host a BS each and the Data Mining Web Service and other two 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 100 requests to the CS in intervals of 2 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. The development of the prototype has allowed assessing the feasibility of implementation, providing a flow of execution from the Client to the GMM access point, and from the Complex Services to the Basic Services, in the sequence depicted in Figure 56.

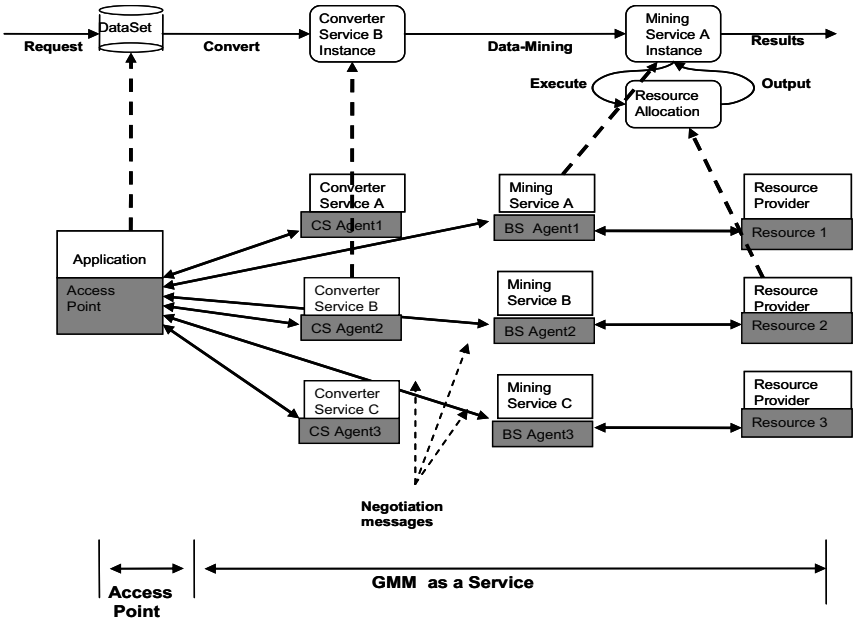

Figure 56. Flow from Client request till Basic Services trading resources.

In general the performance results have been promising in the sense that they show coherent self-oganized behaviours in the bargaining agents. This includes coherent reaction to offer/demand variations, varying computational size of services to be executed and the effect of background loads in the Grid. Improved control of such behaviours could be achieved with further refinements of both multiagent coordination protocols (i.e. the catatllactic agents temsleves and a measurement infrastructure incorporating latest advances in complexity management. Another interesting property which could not be assessed in the prototype (given the material limitations of the number of physisical nodes available) is that of scalability. However, the results from the simulator on improved scalability of decentralized catallactic agents over centralized approaches offers a good insight on the "nice" scalability properties of the catallactic agents. The results for the experiments with Catallactic agents in the prototype are covered in section 3.8.3.

5 Discussion of results

Our discussion utilizes the results from simulating the centralized and catallactic allocation approach and the lessons learnt implemeting the catallactic approach into the prototype.

In general, the CATNETS project has achived the following exploitable outcomes:

- a specification of decentralized and centralized market protocols,
- a simulator which is able to compare the centralized and the decentralized allocation approach,
- a framework for economic analysis of centralized and decentralized economic mechanisms,
- a prototype of the catallactic mechanism integrated into different applications, and
- the Grid Market Middleware (GMM) which feeds into other FP6 projects like SORMA.

In Section 5.1, the statements of the simulator are summarized. Section 5.2 discusses the statements of the prototype, and Section 5.3 gives a short conclusion on the applicability of the catallactic approach to application layer networks. Finally, further research properties for Hayek's Catallaxy are presented in Section 5.4.

5.1 Statements for simulator

The simulator allows carrying out experiments up to a few thousands of nodes. However, very large-scale experiments with several thousands of nodes consume lots of main memory. The automated scenario generator is tested with a scenario up to 10000 nodes and 100000 agents. This requires a machine with 8GB of main memory. The simulator is able to read a scenario of this size and to start the simulation with 1000000 requests. But, it takes up to weeks to finish such very large simulation runs. Accessorily, the memory management of the Java Virtual Machine has to be optimized by modifying heap size and memory consumption parameters, and the Linux file system parameters have to be changed to be able to handle such large simulation runs. Therefore, no very large scale experiments are evaluated because the computational resources are needed to simulate the smaller experiments for the scenarios of this deliverable.

In the largest scale simulations, we could only apply the Catallactic mechanism, but not the centralized mechanism because components of the centralized auction implementation consume a large number of resources depending on the size of the scenario. We observe that the Catallactic mechanism could principally achieve very good service allocation. However, this mainly depends on configuring its parameters correctly.

The development of the simulator and the experiments show a high complexity of the agent strategy in catallactic system. Agein, the performance depends on numerous parameters which are not easily configurable appropriately.

In the simulations, a number of advantages and disadvantages of the decentralized mechanism compared to the centralized catallactic mechanism have been identified:

- **Number of messages:** The centralized mechanism used a significantly smaller number of messages for communication. The bargaining of the catallactic mechanism had a significantly higher cost in terms of messages exchanged. The number of messages in the implemented auctions remains always the same for one allocation, whereas the number of messages varyies along the iterative negotiation steps in the decentralized allocation approach. By limiting the number of negotiation rounds, a lower number of messages can be achieved. For example, a high price step value together with a high aquisitiveness decreases the total number of messages for an agreement.

- **Negotiation attributes:** In the catallactic mechanism, only single-attribute negotiations could take place, whereas the auctioneer mechanism of the centralized approach could handle multi-attribute negotiations. In our particular case, this limitation was imposed by the agents` strategy implementation. The learning and decision algorithms were implemented to cover only single-attribute negotiations. An extension to multi-attribute, however, has several difficulties, like matchmaking of multi-attribute services in a decentralized search.

- **Negotiations:** The implementation of the agents was based on a model in which each Catallactic agent could only be involved in one negotiation at a time. This model poses serious limitations on performance. A more efficient model with parallel negotiations isn't implemented due to its additional complexity. The same limitation holds with the centrlized allocation approach. Parallel bidding is not supported.

The final statement "Catallaxy is more / less efficient than central mechanisms" could not be obtained from the experiments in general. However, there is a catallactic strategy configuration which achieves equal or better performance in terms of equal or better social utility values than the centralized allocation approach of the simulated scenarios. The following insights have been accomplised:

- The Catallactic mechanism needs to be improved in terms of messages needed. A strategy proposal using only one negotiation round is presented in deliverable D2.3. The total number of messages could be reduced be increasing the price step value in the current strategy implementation.

- The configuration complexity in terms of the number of parameters compared with other economic mechanisms is very high in the Catallactic mechanism. On the one hand, this flexibility enables to find a good parameter configuration in all

evaluated scenarios. On the other hand, there is no single configuration which can be applied in most scenarios.

- Comparison with centralized mechanisms is difficult since the performance does depend on each catallactic agents` strategy in the catallactic scenarios. In the centralized case, the auctioneer`s decisions only depend on the incoming supply / demand messages. Catallactic agents follow a heuristic strategy whereas the centralized auctioneer implements a mathematical algorithm with theoretical foundations (state-of-the-art matching mechanism).

- Advantages of the centralized mechanism originate in the significantly lower number of messages needed and the fact that the catallactic case supports only single-attribute negotiations can take place, whereas the auctioneer mechanism can handle multi-attribute negotiations. This limitation was imposed by the agents` strategy implementation. The learning and decision algorithms were implemented to cover only single-attribute negotiations. For this project a model was used in which each agent could only be involved in one negotiation at a time. This model posed serious limitations on performance – for a more efficient system the option of parallel negotiations would have to be implemented. This model also results in a very restrictive blocking policy of the agents.

- Implementation aspects of the simulator were underestimated. More time was spent on testing of the simulator tools and the implemented agents. Therefore, the co-allocation implementation and the shared resource model have still experimental status. No meaningful simulations are evaluated with these features.

- Analyzing the trading agents in the decentralized strategy, two dominant trading strategies lead to profit (positive fitness) in a bilateral negotiation. Either the agents make often concessions with small steps or the agents follow a strategy with a low concession rate together with a high step size. Weighting the last agreements with 60% seems a good value for the successful agents.

- No clear picture evidences the applied service and resource distributions in scenarios with the catallactic strategy. A placement of resource on good connected nodes results in better social utility values. This is an argument for organizing computing centers in a centralized way. Uniformly distributed service and resources lead to good availability of the specific service and resource. But, higher deviations from the mean values could be observed than in other experiments. The competition for several service and resource increases the social utility index and the system’s loss.

- In absolute numbers, the decentralized allocation approachis able to achieve a social utility index value of around 0.4. This value is 20 % better then the measured value for the centralized allocation approach in the comparison scenario. Agein, this indicates the strong influence of the configuration on the allocation performance of the catallactic approach.

- In the evaluated scenarios, the decentralized allocation approach shows higher allocation rates than the centralized allocation approach. The centralized method sacrifices a higher allocation rate the economic outcome of the auction.

- Varying discovery timeouts increase the availability of the sellers and point at successful refinement of the decentralized search strategy.

- The implemented decentralized bargaining protocol is error-prone which exhibits the failure experiments. At the time of development, the main focus was more at a sound and functional messaging structure and implementation than a failure-resistant bargaining protocol. More effort is needed for improving the bargaining protocol.

- The metric pyramid and the selected metrics enable the analysis of complex application layer networks. Changes in the measured metrics are mapped very well to the upper layers of the pyramid. The aggregation steps keep the characteristics of the raw data and indicate different allocation performances in the evaluated scenarios. The general applicability of the metrics pyramid offers a broad evaluation of resource allocation approaches in future application layer networks. However, the metrics pyramid exposes limitiations. The number of observations and the agent population should be evaluated in parallel to verify the expressiveness of the aggregated indexes.

- The decentralized allocation performance does not decrease in increasing network sizes. The hop count and discovery timeout parameters control the accessible area of the network for the decentralized allocation approach. Only a subset of (theoretically) available service and resources can be selected as trading partners. This helps achieving good social utility index values in large networks.

- Varying bandwidth decreases the system loss for the decentralized allocation algorithm. The bandwidth controls the set of available sellers. Not reachable seller can provide their service to another buyer which is located closer the requestor. Also, the set of unique trading agents increase. A more distributed behavior is observed. However, too low bandwidth increases the system's loss significantly.

- The hop count parameter which can be seen as a time-to-live parameter in P2P networks shows significant influence on the decentralized allocation approach. The following rule is derived from the experiments: The higher the service density is, the lower the hop count parameter should be selected.

- Randomizing the intital genotype values lead to lower performance of the decentralized allocation approach because the agents need time to adapt to the dominant strategies described above. Compared to simulations runs with one predefined genotype for all agents, a siginifcant lower performance is measured.

- A crucial part of the catallactic strategy is the adaptation of the negotiation intervals for the the next negotiation which depends on the market price estimation. The selected dynamic strategy exhibits good performance in all simulation scenarios. It is expected that an improved adaptation strategy can lead to better performance in terms of a decreasing message number and better allocation times.

5.2 Statements for prototype

The implementation has been achieved in terms of a prototype. This implementation can be considered as successful. For a production-quality implementation of the Catallactic approach, however, several identified limitations would need to be solved.

- Handling of messages: In order to handle the large number of messages exchanged, an efficient implementation of such a component is needed.

- Multi-attribute: Scalable matchmaking of multi-attribute distributed objects is not completely solved.

- The flexibility of the agent implementation in terms of handling parallel negotiations would need to be improved.

- Particular components of the architecture, like the Catallactic Access Point (CAP) need to be able to handle a high load in terms of messages.

- We have designed, implemented and deployed a fully decentralized prototype incorporating emergence and self-organization using state of the art tooling. The GMM implementation has been proven as useful in several applications. The use of the Catallactic middleware has been shown by two applications: COVITE being available as Grid Service and Data Mining tools given as Web Services. We have deployed this prototype in several scenarios.

- 1CS vs. 2BS scenario is working well. Sufficient negotiations finshed.

- 2CS vs. 4BS scenario fewer negotiations executed due to implemented model (blocking policy because of not allowed parallel negotiations)

- 3CS vs. 6BS scenario suffers the same problem (worse than in 2vs4 scenario) - problem of catallactic model that was implemented

- Important trouble during prototype calibration has come from the requirement deal with decentralized decision makers (the trading agents) in a real, networked,

infrastructure. This issues are related to the broader problematic of mastering complexity

- The open issue with the catallactic-enabled applications is to achieve improved system control, in the form of more predictable outcomes out of the emergent properties of the markets.

- The middleware tool produced by the project (GMM) as well as the prototype implementations built on it can be the seed of further advances in the endeavour of egeneering with complexity in distributed systems, paving the way for more prototype implementations and providing a valuable experience.

5.3 Results on the applicability of the Catallactic approach

In the simulations for the small scale scenarios, the centralized mechanism offers better performance than the decentralized catallactic mechanism. The centralized mechanism is able to handle small or medium scenarios very well. The highly efficient clearing mechanism and the ability to handle bundled goods outperform the catallactic approach.

For large scale scenarios, only the Catallactic mechanism could be simulated. However, the message costs are very high. At least a redesign to reduce the number of messages would be needed. Simulations with a reduced number of messages (using high priceStep values) reach a total social utility of 0.4 which is better then the measured social utility of 0.5 of the centralized allocation approach. This emphasizes the importance of the correct catallactic strategy setup.

A Proof-of-Concept implementation of a Catallactic enabled application and the grid market middleware has been achieved. A production-quality implementation would require substantial improvements of several components like the communication layer and the distributed matchmaking.

5.4 Further research on properties of Catallaxy applied to computer networks

From a computational and engineering perspective, Hayek's Catallaxy is successfully applied to computer networks in the CATNETS project. Explicit specifications of the Catallaxy for technical systems should encompass these agent features:

- Adaptivity: Agents learn from their own experience and from previous agreements, there is genetic recombination, mutation and selection, and agents are reactive and opportunistic being able to adapt their goals to local unpredictable and evolving environments. The current catallactic strategy implemented the first two adaptivity topics. The third issue gives further research prospects. More

parameters could be taken into account for dynamic adaptation by the agent itself instead of being predefined at experiment start.

- Autonomy and initiative: The complex service handles the task to be executed. If it finds an opportunity (a seller which fits the requirements) – proactively without the direct control of the user application – the agent delegates not only a specific task but also an objective to bring out in any way; the agent will find its way on the basis of its own learning and adaptation, its own local knowledge, its own comtetence and reasoning, problem solving and discretion. The results of the negtoation analysis demonstrate the successful application of the catallacatic strategy to the CATNETS scenario. Agents gain profit depending on their own learning and adaptation capabilities, and their local knowledge. Of course, not all agents are successful in making profit. They haven't the knowledge for finding satisficing goals. Dissatisfaction is related to the agents' goals, which Hayek calls "the conscious purpose". The agents' level of wealth, knowledge and consumption determines the "conscious purpose" as a function of which the agent will be more or less satisfied, and more or less responsive to existing circumstances. If the agent knows only other agents with low fitness and is unaware of what else is possible, the agent will act within its local environment and isn't able to escape from it. There could be agents is the application layer network, which never gain profit. Further research should address finding a lower bound for the agent success for the catallactic strategy.

- Distribution and decentralization: In CATNETS, the multi-agent systems are open and decentralized. As the results of the simulations runs evidence, it is neither established nor predictable which agents will be involved in trades. There are no regional analysis tools of the agents and its trading partners available. Placing resource on good connected and varying bandwidth on the links increase the distribution and decentralization of the multi-agent system. Further experiments are needed to find out more rules for increasing the agents' distribution and decentralization. The execution times of the tasks are assumed to be constant in CATNETS. Agents may remain open during execution by means of being reactive to incoming inputs and to the dynamics of its internal state (for example, resource shortage or preference change). Further research on the Catallaxy could take this into account increasing the reactivity of the agents. The workflow simulations with a complex service requesting a sequence of basic service give prospects for further analysis, because nobody in the system knows the complete plan. Even the complex services don't know to which basic service a task on second or third place could be delegated. Policies are needed to ensure that the task of long workflows could still be executed because longer workflows come along with an increased failure probability.

The products traded on the service market are homogeneous products which are completely standardized. The experiment runs of the lessons learnt from CATNETS show, the implemented centralized and decentralized allocation approaches handle these products very well. The implemented Hayek's Catallaxy can even outperform the centralized auctions. The introduction of heterogeneous products enables a more realistic

service picture. Also the users or applications of the catallactic market profit from heterogeneous services. They can specify their workflows with more details like service quality levels, priorities, etc. This leads to a broader applicability of the Catallactic strategy in new application domains and real world user requests.

As mentioned above, CATNETS assumes an open application layer market implemented by an open multi-agent system. The traded quality of the products in terms of execution time is always constant. No service quality changes are taken into account. Further research prospects could extend the CATNETS model with changing service quality which is required by future interactive Grid and P2P application based on service-oriented architecture (see Case Study in deliverable D3.3). Additionally, shared resource models have to guarantee certain quality levels to users. Hayek's concept of a "spontaneous order" could be eroded by cheating agents in such evironments. There is no institution which forces the agents to be honest. The Catallaxy could benefit by the introduction of electronic institution and concepts from social control like reputation mechanisms. This comes along with a change from single attributive decision making to multi-attributive decision making and resoning. The agents have to decide how much risk they agree to and how they should adapt their preference structure in case of risky services and resources. Besides using social concepts, these risky assets can also be handled by economic risk management. Concepts from risk transfer like insurances or risk mitigation like portfolio optimization could help to reduce the financial risk of the agents. It would be interesting to see how the influence of the social and economic approaches influences the CATNETS metrics paramid for evaluation of Hayek's Catallaxy.

Future hardware developments will soon make possible the construction of very large scale (one million of agents and above) models that obviate the need for representative agents. Artificial agent communities and economies of such scale need more research to overcome the lack of understanding of realistic behaviour of agents and institutions. Hayek's Catallaxy is one important concept to help understanding such complex and dynamic agent communities. We identified some future research prospects in the field of artificial economies:

- The markets could benefit from the introduction of new agent roles. Intermediate traders which buy and sell basic services increase the liquidity of the markets. More trading on the markets will lead to more exchanged price signals. The agents' learning and reasoning components could gain better price estimations.
- The price signals which are exchanged on the markets should be honest and unaltered. An independent monitoring institution could ensure trusting price signals. Traders pay for this market monitoring service. A loss of confidence would force the traders to select another trading partner.

Further research and impact of the CATENTS project is listed in the dissemination and use plan and the technology implementation plan. Several national and international projects profit from the results gained by the CATNETS project. The reader is referred to those deliverbales for further information.

6 Conclusions

In Application Layer Networks like future Grid and P2P networks, optimal resource allocation has to be carried out in two dimensions. One is the maximization of the utilization of technical resources. The aim in this dimension is - independently of the economic incentive structure - to carry out a load balancing on heterogeneous and distributed computational resources. This guarantees that all resources are used and no resources are "wasted" while being idle. In contrary to this allocation paradigm, the economic resource allocation is aligning the deployment of resources along the economic utility of the individual Grid nodes. Mechanisms like multi-attribute combinatorial exchanges enable an incentive compatible, efficient, individual rational and computational tractable way of allocating these resources. This may imply that resources stay idle in case the willingness to pay of the resource requester does not reach the reservation price stated by the resource owner. This effect is demonstrated by the experiment runs: The centralized resource allocation approach shows lower allocation rates than the catallactic allocation method. However, it enables an economically sound construction of allocation for Grid resources.

Within the CATNETS project, two dimensions of such markets have been investigated: One is the application of decentralized - catallaxy-based - market mechanisms, which uses flooding for resource discovery and an iterative bilateral bargaining protocol for negotiation of resources. The alternative is the application of classic institutional - or centralized central - market mechanisms. If properly designed, markets implement an economical efficient allocation of resources. Such multi-attribute multi-unit mechanisms to ALN/Grid allocations have been developed as a benchmark. They provide an allocation for Grid resources up to a certain size. Mechanisms are developed that enable a two-tiered allocation of Grid resources, which fulfill most of the above-mentioned desiderata. In the first tier of these markets, service consumes, who may not have a clue of what kind of computational resources they need, can trade with service providers about their service needs. These service providers themselves then act on a second market tier - the resource market - where they purchase the resources they need in order to carry out the services. These two markets are interrelated through the price that is determined in the first tier. State of the art in this research field is that incentive compatible, allocative efficient and individual rational allocation mechanisms are identified. However, these mechanisms are very complex to compute for large number of market participants (they are NP-complete) and hence not applicable on large-scale setups while the decentralized market mechanism still achieves performant results in large settings.

The comparison of the two approaches shows that:

- The centralized mechanism used a significantly smaller number of messages for communication. The bargaining of the catallactic mechanism had a significantly higher cost in terms of messages exchanged. The number of messages in the

implemented auctions remains always the same for one allocation, whereas the number of messages varyies along the iterative negotiation steps in the decentralized allocation approach. By limiting the number of negotiation rounds, a lower number of messages can be achieved.

- In the catallactic mechanism, only single-attribute negotiations could take place, whereas the auctioneer mechanism of the centralized approach could handle multi-attribute negotiations. In our particular case, this limitation was imposed by the agents` strategy implementation. The learning and decision algorithms were implemented to cover only single-attribute negotiations.

- The implementation of the agents was based on a model in which each Catallactic agent could only be involved in one negotiation at a time. This model poses serious limitations on performance. A more efficient model with parallel negotiations isn't implemented due to its additional complexity. The same limitation holds with the centrlized allocation approach. Parallel bidding is not supported.

- The potential of Catallaxy as resource allocation mechanism has been motivated by observing its usage in many real life situations. From the project results, we have noted that the complex models which are behind this usage, however, have not shown to be easily portable into artificial systems.

- The use of Catallaxy in daily live situations requires very sophisticated capabilities, which are not easily reproducable by technical implementations. Providing each artificial agent by the "intelligence" needed to work in such a resource allocation approach has currently a high cost and needs an expert user.

- The contact with the system, the information dissemination and lookup, the interaction with other participantes require very challenging technical solution. It is considered difficult to provide technical solutions based on standard toolkits which could provide reliable a funtionality without limitations in large-scale scenarios and underrealistic conditions.

- The scenarios which we could consider have shown to be successfully realizable with less complex solutions than Catallaxy like a centralized approach based on auctions. The limitations of the centralized solution have not been identified in the scope of the requirements. Potential bottlenecks like the centralized components appear to have easier solutions for overcoming them than applying Catallaxy.

- We cannot exclude the possibility that particular scenarios which might arise from future distributed applications could provide circumstances in which Catallaxy remains an interesting option for providing a solution. Providing the building blocks of Catallaxy in a useable way, achieve a feasible configuration by the users, however, would ease any practical application of the approach.

- A final statement on the performance of the Catallaxy approach to centralized auction approach is difficult to obtain. A main problem lies in the technical aspects of the implementation. Catallaxy works best in large scale scenarios, but a sufficient simulation for Catallaxy needs larger technical resources. On the other hand, the simulation time needed for the centralized approach increases dramatically with growing simulation size.

- From an implementation perspective, even the moderate complexity of the heuristic bargaining strategy leads to a noticeable variance of the simulation results, when compared with the predictable results of the auctioneer's algorithm. The calibration of the simulation and a working prototype became an important task in the CATNETS project.

- With regard to the Grid market parameters, we have achieved various possibilities for adaptation to real world settings. Virtualized resources and resource bundles are supported.

- The final statement "Catallaxy is more / less efficient than central mechanisms" could not be obtained from the experiments in general. However, there is a catallactic strategy configuration which achieves equal or better performance in terms of equal or lower social utility values (that means better score for social wellfare) than the centralized allocation approach of the simulated scenarios.

The recommendation for future research in this domain is threefold:

1. More research effort should be devoted to the combination between the communities that do research in the technical allocation and conceptualization of application layer networks as well as the economic allocation in future Grid and P2P networks. Both sides can - while commercializing the application layer networks - not continue without the other.

2. Besides technical standardization, efforts for the development of economic standards and interaction schemes, e.g. based on Web Services or other concepts from Service Oriented Architectures, should be fostered for the practical utilization of future e-Infrastructures and e-Science.

3. Efforts for starting real-life pilots for Grid/P2P business models and Grid/P2P markets should be fostered, where researchers from computer science, economics and business administration commonly work on dynamic, economically sound and vertically integrated business concepts for the dynamic utilization of application layer networks and other e-Infrastructures.

References

[ACC+05] O. Ardaiz, P. Chacin, I. Chao, F. Freitag, L. Navarro . An Architecture for Incorporating Decentralized Economic Models in Application Layer Networks . International Journal on Multiagent and Grid Systems. Special Issue on Smart Grid Technologies

[ACC+05] O. Ardaiz, P. Chacin, I. Chao, F. Freitag, L. Navarro . An Architecture for Incorporating Decentralized Economic Models in Application Layer Networks. International Journal on Multiagent and Grid Systems. Special Issue on Smart Grid Technologies. Volume 1, Number 4 / 2005, pp. 287 – 295.

[AAE+06] O. Ardaiz, P. Artigas, T. Eymann, F. Freitag, L.Navarro, M. Reinicke --The Catallaxy Approach for Decentralized Economic-based Allocation in Grid Resource and Service Markets", Special Issue on Agent-based Grid Computing, International Journal of Applied Intelligence. 2006.

[AH00] Adar, E. and B. Huberman: 2000, 'Free Riding on Gnutella'. First Monday 5(10).

[BCC+07] Brunner R., Chao I., Chacin P., Freitag F., Navarro L., Ardaiz O., Joita L., and Rana O.F. Assessing a distributed market infrastructure for economics-based service selection?, accepted to the International Conference on Grid computing, High-Performance and Distributed Applications (GADA'07), Vilamoura, Algarve, Portugal, Nov 29 - 30, 2007.

[CAS+07] Chao I., Ardaiz O., Sangüesa R., Joita L., and Rana O.F. Optimizing Decentralized Grid Markets through Group Selection?. Submitted to the Workshop on Economic Models and Algorithms for Grid Systems, in conjunction with The 8th IEEE International Conference on Grid Computing (Grid 2007), Austin, Texas, USA, September 19 - 21, 2007.

[CBF+07] Chao I., Brunner R., Freitag F., Navarro L., Chacin P., Ardaiz O., Joita L., and Rana O.F. (2007). Decentralized Grid Market Infrastructure for Service Oriented Grids, accepted for the 2nd review process to the Special Issue (01/2008) on Service-Oriented Architectures and Web Services, in Journal WIRTSCHAFTSINFORMATIK (Business Informatics), Germany, 2008.

[CUB+05] H. Czap, R. Unland, C. Branki and H. Tianfield (editors), pp 18 - 34. ISBN: 1-58603-577-0, IOS Press. Proceedings of the International Conference on Self-Organization and Adaptation of Multi-agent and Grid Systems (SOAS 2005), Glasgow, Scotland, UK

[Del05a] O. Ardaiz, P. Chacin, I. Chao, J.C. Cruellas, F. Freitag, L. Joita, M. Medina, L. Navarro, O.F. Rana, M.Valero. CATNETS Deliverable WP 3, Year 1, 2005.

[Del05b] Michele Catalano, Gianfranco Giulioni, Werner Streitberger, Michael Reinicke, Torsten Eymann. CATNETS Deliverable WP 4, Year 1, 2005.

[Del06a] Zini, F. et al. CATNETS Deliverable WP 2, Year 2., 2006.

[Del06b] Michele Catalano, Gianfranco Giulioni, Werner Streitberger, Michael Reinicke, Torsten Eymann CATNETS Deliverable WP 4, Year 2, 2006.

[Del07a] Zini, F. et al. CATNETS Deliverable WP 2, Year 3., 2007.

[Del07b] O. Ardaiz, P. Chacin, I. Chao, F. Freitag, L. Joita, L. Navarro, O.F. Rana. CATNETS Deliverable WP 3, Year 3, 2007.

[DUA04] 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

[ECE05] http://cfpm.org/~david/ece2005/

[FCC+07] F. Freitag, P. Chacin, I. Chao, R. Brunner, L. Navarro, O. Ardaiz, Performance Measuring Framework for Grid Market Middleware. Accepted in 4th European Performance Engineering Workshop (EPEW 2007), Berlin, Germany, September 2007.

[Fri91] Friedman, D. The Double Auction Market Institution: A Survey. In The Double Auction Market - Institutions, Theories, and Evidence, D. Friedman and J. Rust, Ed. Cambridge MA, Perseus Publishing, 1991, pp. 3–26.

[Gol93] Goldberg, D.: 1993, Genetic Algorithms in Search, Optimization and Machine Learning. Reading MA, Addison Wesley.

[Hay89] Hayek, F., W. Bartley, P. Klein, and B. Caldwell: 1989, 'The collected works of F.A. Hayek'. University of Chicago Press.

[JRC+05] L. Joita, O. Rana, P. Chacin, I. Chao, F. Freitag, L. Navarro, O. Ardaiz. Application Deployment using Catallactic Grid Middleware. 3rd International Workshop on Middleware for Grid Computing (MGC 2005), co-located with Middleware, Grenoble, France, 2005.

[JRC+06] L. Joita, O. F. Rana, P. Chacin, I. Chao, F. Freitag, L. Navarro, O. Ardaiz. Application Deployment on Catallactic Grid Middleware. IEEE Distributed Systems Online, vol. 7, no. 12, 2006, art. no. 0612-oz001.

[JRF+07] L. Joita, O. F. Rana, , F. Freitag, I. Chao, P. Chacin, L. Navarro, O. Ardaiz. A catallactic market for data mining services. Future Generation Computer Systems, Volume 23, Issue 1, 1 January 2007, pages 146-153.

[KR95] Kagel, J. and A. Roth: 1995, The handbook of experimental economics. Princeton University Press.

[PT98] 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

[Pru81] Pruitt, D.: 1981, 'Negotiation behavior'. Organizational and occupational psychology. NewYork: Academic Press.

[RHJ+05] Omer Rana, Jeremy Hilton, Liviu Joita, Pete Burnap, Jaspreet Singh Pahwa, John Miles, W. Alex Gray, -- SECURE VIRTUAL ORGANISATIONS: Protocols and

Requirements", at Information Security Solutions Europe (ISSE) 2005, 27-29 September 2005, Budapest, Hungary, ISBN 3-8348-0011-2, p. 422-432, 2005.

[RZ94] Rosenschein, J. S.; Zlotkin, G.: 1994, 'Rules of encounter - designing conventions for automated negotiation among computers'. MIT Press, Cambridge.

[Sch07] Schnizler, B. Resource Allocation in the Grid -- A Market Engineering Approach, Universitätsverlag Karlsruhe, 2007.

[SEV+07] Streitberger W., Eymann T., Veit D., Catalano M., Giulioni G., Joita L., Rana O.F. 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.

[SOR07] http://www.iw.uni-karlsruhe.de/sormang/

[ST98] Smith, R. E. and N. Taylor: 1998, 'A Framework for Evolutionary Computation in Agent-Based Systems'. In: Proceedings of the 1998 International Conference on Intelligent Systems. ISCA Press.

[Tes97] Tesfatsion, L.: 1997, 'How economists can get alife'. In: The Economy as a Evolving Complex System II. Arthur, W.B. and Durlauf, S. and Lane, D.A. (Hrsg.), pp. 533-564.

[TU05] H Tianfield and R Unland (2005). Towards self-organization in multi agent systems and Grid computing. Multiagent and Grid Systems – An International Journal 1(2), October 2005, pp. 89-95

[Var94] Varian, H. R.: 1994, Mikrokonomie. Oldenbourg.

[WH05] T. DeWolf, and T. Holvoet, Towards a Methodolgy for Engineering Self-Organising Emergent Systems, In Self-Organization and Autonomic Informatics (I), Volume 135 of Frontiers in Artificial Intelligence and Applications.

Annex A – CATNETS Repositories Settings

The metric collector script builds on vxargs to perform remote ssh connections in parallel to all the nodes involved in the system, gathering all the raw data into a central repository:

```
------------------- start of collecMetrics script ---------------------
---

#collectMetrics.sh
#author: Isaac Chao; ichao@lsi.upc.edu
#collects metrics and logs from the middleware in a set of machines
listed in nodeListBS.txt and nodeListCS.txt
#metrics and logs from an experiment are stored in a file
metricAndLogsMiddleware_$LABEL

#edit YOURPATH to change the remote nodes GMM path (this must be
identical to  the PATH used for GMM deployment)
#edit RESULTSPATH in the case you need the experiment results to be
stored in a different path

# to run please execute ./collectMetrics yorName
# this will identify the result of your experiment unambigously and
store it in RESULTS_HISTORY_PATH

YOURPATH=$HOME/GMM
RESULTSPATH=$HOME/results
RESULTS_HISTORY_PATH=$HOME/resultsHistory
USER="user"
TYPE="Catallactic"

#variables
LABEL=$(date +%Y-%m-%d-%H-%M)"_"$TYPE"_"$USER


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

# create directories to collect metrics
mkdir $RESULTSPATH
mkdir $RESULTSPATH/metrics
mkdir $RESULTSPATH/logs
mkdir $RESULTSPATH/metrics/negotiation_start
mkdir $RESULTSPATH/metrics/negotiation_request
mkdir $RESULTSPATH/metrics/negotiation_end
mkdir $RESULTSPATH/metrics/strategy_metric
mkdir $RESULTSPATH/metrics/executionTime


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

```
echo " "
echo "collecting negotiation_start metric from remote nodes via
vxargs... "
./vxargs -p -a nodeList.txt scp
{}:$YOURPATH/middleware/negotiation_start.txt
$RESULTSPATH/metrics/negotiation_start/BS{}_negotiation_start.txt

echo " "
echo "storing negotiation_start metrics from:
$RESULTSPATH/metrics/negotiation_start/ into file:
$RESULTSPATH/metrics/negotiation_start/negotiation_start.txt... "
cat $RESULTSPATH/metrics/negotiation_start/* >>
$RESULTSPATH/metrics/negotiation_start/negotiation_start.txt


echo " "
echo "collecting negotiation_request metric from remote nodes via
vxargs... "
./vxargs -p -a nodeList.txt scp
{}:$YOURPATH/middleware/negotiation_request.txt
$RESULTSPATH/metrics/negotiation_request/BS{}_negotiation_request.txt

echo " "
echo "storing negotiation_request metrics from:
$RESULTSPATH/metrics/utilization/utilization.txt
RESULSPATH/metrics/negotiation_request/negotiation_request.txt ... "
cat $RESULTSPATH/metrics/negotiation_request/* >>
$RESULTSPATH/metrics/negotiation_request/negotiation_request.txt


echo " "
echo "collecting negotiation_end metric from remote nodes via vxargs...
"
./vxargs -p -a nodeList.txt scp
{}:$YOURPATH/middleware/negotiation_end.txt
$RESULTSPATH/metrics/negotiation_end/BS{}_negotiation_end.txt

echo " "
echo "storing negotiation_end metrics from:
$RESULTSPATH/metrics/negotiation_end/ into file:
$RESULTSPATH/metrics/negotiation_end/negotiation_end.txt... "
cat $RESULTSPATH/metrics/negotiation_end/* >>
$RESULTSPATH/metrics/negotiation_end/negotiation_end.txt


echo " "
echo "collecting strategy_metric  metric from remote nodes via
vxargs... "
./vxargs -p -a nodeList.txt scp
{}:$YOURPATH/middleware/strategy_metric.txt
$RESULTSPATH/metrics/strategy_metric/BS{}_strategy_metric.txt

echo " "
echo "storing strategy_metric metrics from:
$RESULTSPATH/metrics/strategy_metric/ into file:
$RESULTSPATH/metrics/sellsuccess/strategy_metric.txt... "
```

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

echo " "
echo "collecting executionTime metric from remote nodes via vxargs... "
./vxargs -p -a nodeListCS.txt scp {}:$YOURPATH/executionTime/*
$RESULTSPATH/metrics/executionTime/

echo " "
echo "storing price metrics from: $RESULTSPATH/metrics/executionTime/
into file: $RESULTSPATH/metrics/executionTime/executionTime.txt... "
cat $RESULTSPATH/metrics/executionTime/* >>
$RESULTSPATH/metrics/executionTime/executionTime.txt


echo " "
echo "collecting  BS_logs from remote nodes via vxargs into file:
$RESULTSPATH/logs/log_BS{}.txt
 ... "
./vxargs -p -a nodeListBS.txt scp {}:$YOURPATH/middleware/log.txt
$RESULTSPATH/logs/log_BS{}.txt


echo " "
echo "collecting CS_logs from remote nodes via vxargs  into file:
$RESULTSPATH/logs/log_CS{}.txt
 ... "
./vxargs -p -a nodeListCS.txt scp {}:$YOURPATH/middleware/log.txt
$RESULTSPATH/logs/log_CS{}.txt


#storing all metrics in a single folder
echo " "
echo "storing  all metrics txt files in a single folder:
$RESULTSPATH/metrics  ... "
cp $RESULTSPATH/metrics/negotiation_start/negotiation_start.txt
$RESULTSPATH/metrics/negotiation_request/negotiation_request.txt
$RESULTSPATH/metrics//negotiation_end/negotiation_end.txt
$RESULTSPATH/metrics/strategy_metric/strategy_metric.txt
$RESULTSPATH/metrics/executionTime/executionTime.txt
$RESULTSPATH/metrics


#METRICS TAR:
# includes config of the experiment, defininng it unambiguosly
# includes the vcargs output of all controller script  executions
(runExperiment.sh stopExperiment.sh clenAllNodes.sh)
# includes all metrics and logs from remote nodes (BSs and CSs)
# labels unamboguosly the experiment with user and date
echo " "
echo "tar all the results into a single experiment file:
$RESULS_HISTORY_PATH/metricsAndLogsMiddleware_$LABEL.tar  ..."
tar -czf $RESULTS_HISTORY_PATH/metricsAndLogsMiddleware_$LABEL.tgz


BSHostingConfig.properties CSHostingConfig.properties
$HOME/outputVxargs CatallacticBSHostingConfig.properties
```

```
CatallacticCSHostingConfig.properties complexService.properties
basicService.properties resourceAgent.properties strategy.conf
learning.conf


------------------- end of collecMetrics script ----------------------
```

Annex B – Matlab scripts main function behavior of the scripts for the analysis of decentral and central behaviour

In order to access to agent adatabase an example could be useful. The command below access the structure test where are stored all the data experiment. To accessa t a particular experiment select an experiment label as T1184664821646. To do the same for an agent select CSA1Site8, and then one is ready to read the database content for the satisfaction metric :

>> test.T1184664821646.CSA1Site8.Satisfaction

ans =

0.2798
0.6084
1.0000
0.9018
0.5906
0.8965
0.6853
0.6917
0.8454
….

Metric	CSagent	BSagent	RSagent
Allocation_rate	X	X	X
Satisfaction	X	X	X
Allocation_Time	X	X	
Provisioning_Time	X		
Distance	X	X	X
Latency	X	X	X
Usage	X	X	
Messages	X	X	X

Both for central and decentral scenario agent metric are collected as follows:

Below one can find a description and motivation for the schema in the table above:

Allocation Rate:

CS: It is provided by the simulator as the ratio accepts / requests
BS: It is evaluated by the scripts offline. Accepts and Rejects are selected from accepts.txt and rejects.txt files
RS: It is evaluated by the scripts offline. Accepts and Rejects are selected from accepts.txt and rejects.txt files

Satisfaction:

CS: It is evaluated by the scripts offline. Selection of agent satisfaction for each negotiation ended successfully
BS: It is evaluated by the scripts offline. Selection of agent satisfaction for each negotiation ended successfully
RS: It is evaluated by the scripts offline. Selection of agent satisfaction for each negotiation ended successfully

Allocation_Time
CS: Negotiation id selected from CS_BS_mapping. Negotiation id matching from service_usage_mat. Once the pairs of usage are selected, it is evaluated end_time – start _time.
BS: Selection of pairs of end and start usage time from resource_usage_mat. Usage metric is the the difference between end and start.
RS: not available.

Provisioning Time:
CS: selection from complex_service_provisioning_time_mat.
BS: not available
RS: not available

Distance:
CS: selection from distance_mat.
BS: selection from distance_mat.
RS: selection from distance_mat.

Usage:
CS: selection usage metric from service_usage_mat
BS: selection usage metric from resource_usage_mat
RS: not available

Messages:
CS: selection from negotiation messages file
BS: selection from negotiation messages file
RS: selection from negotiation messages file

Agentanalysis2 and Agentanalysis_c

The scripts select the agent present in the experiment runs. They save the agent lists in the variables: `final_CSA, final_BSA, final_RSA .` The accepts and rejects numbers are selected from the accepts.txt and reject.txt files to evaluate the allocation rate for BSA and RSA. (The CSA allocation rate are already available in the complex_service_agent_allocation_rate.txt). The formula is

alloc.rate = accepts / (rejects + accepts) .

For each **CSA** are selected the remaining variables:

Satisfaction.
From `util_satisfaction_service_buyer_decentral.txt` the satisfaction's agent entry are selected using the agent_id.

So for example the lines:

```
M=selcell(util_satisfaction_service_buyer_decentral_mat{k,1}{1,2},final
_CSA{i});
agent.Satisfaction=util_satisfaction_service_buyer_decentral_mat{k,1}{1
,5}(M);
```

save in the `agent.Satisfaction` structure array the satisfaction entries stored in the variables `util_satisfaction_service_buyer_decentral_mat` (k is the experiment number and 5 is the column where are stored the agent id). The exact entry indexes are selected with the `selcell` function, which take as input the 2th column `util_satisfaction_service_buyer_decentral_mat` (where are stored the agent_id) and the `final_CSA{i}` the i-th agent_id. The selcell function behavior is basically a string comparison between an array and a single string.

complex_service_provisioning_time.
From the related file the entry are selected using the agent_id.

Distance
From distance.txt the entry are selected using the agent_id.

Usage
From service_usage file are selected the agent entries. The final observation is usage = end.time_service_usage{i} - start.time_service_usage{i}.

Messages.
The messages are selected as the metric above, from the file negotiation_messages.txt

Allocation time
The entries are selected from the basic_service_Allocation_time.txt and as usage the time are computed as end.time_bas_alloc_time{i} - start.time_bas_alloc_time{i}.

Agent Metrics saving

In the line below:

```
experiment=setfield(experiment,char(CSA{i}),agent);
```

are stored in the structure array experiment the agent metrics. This is done for all the CSagent population.

BSA metrics

As the CSA the main metrics are selected as below:

Satisfaction.
From `util_satisfaction_service_seller_decentral.txt` the satisfaction's agent entry are selected using the agent_id.

Provisioning Time
Not available

Distance
From distance.txt the entry are selected using the agent_id.

Usage
From resource_usage file are selected the agent entries. The final observation is usage = end.time_resource_usage{i} - start.time_resource_usage{i}.

Resource allocation time
as above for the complex_service_agents, this time entries are selected from resource_allocation_time.txt

Messages.
The messages are selected as the metric above, from the file negotiation_messages.txt

Agent Metrics saving

In the line below:

```
experiment=setfield(experiment,char(BSA{i}),agent);
```

the agent metrics are stored in the structure array `experiment`. This is done for all the BSagent population.

RSAgent

For each **CSA** are selected the remaining variables:

Satisfaction.
From `util_satisfaction_service_buyer_decentral.txt` the satisfaction's agent entry are selected using the agent_id.

Provisioning Time
Not available

Distance
From distance.txt the entry are selected using the agent_id.

Usage
Not available

Resource allocation time
Not available
Messages.
The messages are selected as the metric above, from the file negotiation_messages.txt

Agent Metrics saving

In the line below:

```
experiment=setfield(experiment,char(RSA{i}),agent);
```

the agent metrics are stored in the structure array `experiment`. This is done for all the BSagent population.

Annex C– Matlab scripts for simulator analysis

Agent_analysis2 and Agent_analysis_c

The scripts has to select and associate the metrics to the agent, in ordet to biuld a complete database taking into account the metrics selected for the simualtor. In what follow are described the main procedures. Are shown the agentanalysis2 script code because the central counterpart it is identical.

```
H=strfind(accepts_mat{k,1}{1,2},'CSA');                % The command use the strfind function in order to find test
with 'CSA'                                                    % string from the 2 column of the accept matrix in
the k experiment


for i=1:length(H)                                      % The folowing statements are to find the final list,
    if isempty(H{i})==1                                % of the CSAgents
        g(i)=0;
    else
        g(i)=i;
    end
end
l=find(g);
CSAgent=accepts_mat{k,1}{1,2}(l);


clear M H                                               % Once the CSA list is available, eventually replication are
deleted and                                                   % saved in the variable final_CSA
H_com=l;
j=0;
for i=1:length(CSAgent)
    if H_com(i)>0
        j=j+1;
         M=selcell(CSAgent,CSAgent(i));
         final_CSA{j}=CSAgent(i);
         H_com(M)=0;
    end
end

number_agent=setfield(number_agent,'CSA',length(final_CSA)) %% This statement set the field in the structure
number_agent with the                                                    %% number of agent
```

The next lines are to selet and associate the data to the agent:

```
for i=1:length(final_CSA)
   M=selcell(complex_service_agent_allocation_rate_mat{k,1}{1,2},final_CSA{i});  %% A) It selects index of
complexs.alloc.rate
                                                                                  array
```

```
                                                                        %%    where is present the
'final_CSA{i}'

   agent.Allocation_rate=complex_service_agent_allocation_rate_mat{k,1}{1,4}(M); %% B) It gives to the agent structure
the selected                                                                          Allocation Rate
                                                                        %% In what follows are executed the
same steps as in                                                                   A) and B)
                                                                          %% See below for the description of
selcell function

   M=selcell(util_satisfaction_service_buyer_decentral_mat{k,1}{1,2},final_CSA{i});
   agent.Satisfaction=util_satisfaction_service_buyer_decentral_mat{k,1}{1,5}(M);
   M=selcell(complex_service_provisioning_time_mat{k,1}{1,2},final_CSA{i});
   agent.Provisioning_Time=complex_service_provisioning_time_mat{k,1}{1,5}(M);
   M=selcell(distance_mat{k,1}{1,2},final_CSA{i});
   agent.Distance=distance_mat{k,1}{1,5}(M);
   M=selcell(latency_mat{k,1}{1,2},final_CSA{i});
   agent.Latency=latency_mat{k,1}{1,5}(M);
   M=selcell(service_usage_mat{k,1}{1,2},final_CSA{i});                    %% the usage metric evaluation start
here: M is the                                                                    %% array index where one
can find the Final_CSA
   usacom=service_usage_mat{k,1}{1,1}(M);
   o=0;

%% The usage metric selection is more complicated: It is needed to be
%% selected the usage start and end from service_usage_mat variable. Then
%% the following procedure evaluate the difference for each pair of start
%% and end
   if length(M)==1                                                         %% Usually M should be a 2 elements
array. But in                                                                  %% this case length of M =1
and the end value is                                                              %% substitute
with the simulation end time
       usa=simulation_time_mat{k,1}{1,5}(2)-usacom;
   else
       for j=2:2:length(M)                                                 %% Here for each pair of usage
observation (end and                                                                  %% start) the usa
variable it is creatred. On it are                                                          %%
stored the usage time obseravtions
           o=o+1;
           usa(o)=usacom(j)-usacom(j-1);
       end
   end

   if isempty(M)==0
      agent.Usage=usa;
   else
      agent.Usage=[];
   end

   clear usa usacom
   M=selcell(CS_BS_Mapping_mat{k,1}{1,2},final_CSA{i});
   Negotiation=CS_BS_Mapping_mat{k,1}{1,5}(M);

   P=selcell(negotiation_messages_mat{k,1}{1,2},final_CSA{i});
   neg=negotiation_messages_mat{k,1}{1,8}(P);
   agent.Messages=ceil(neg);

   clear neg

%% The allocation time metric selection is more complicated: It is needed to be
%% selected the usage start and end from service_usage_mat variable. Then
%% the following procedure evaluate the difference for each pair of start
%% and end
   P=selcell(basic_service_allocation_time_mat{k,1}{1,2},final_CSA{i});
   n=0;
   for j=2:2:length(P)
      n=n+1;
      csa_alloc_time1=basic_service_allocation_time_mat{k,1}{1,1}(P(j));
      csa_alloc_time2=basic_service_allocation_time_mat{k,1}{1,1}(P(j-1));
      c_alloc_t(n)=csa_alloc_time1-csa_alloc_time2;
   end
   if isempty(P)==0
   agent.Allocation_Time=c_alloc_t;
   else
   agent.Allocation_Time=[];
   end

   clear P
   clear c_alloc_t
%% Final CSA metric storage

   CSA{i}=regexprep(final_CSA{i}, '@', '');
   experiment=setfield(experiment,char(CSA{i}),agent);                   %% This set the experiment structure
with agent data
   clear Negotiation
end
```

Agent_eval, Agent_eval_c

The scripts collects all the agent data and aggregate them in the upper level metric indicators.

The script asks initially the number of experiment to analyze and if the final comparison has to be done (If yes it saves in a folder the main data to compare them with other scenarios). For each experiment and agent the script evaluate the total number messages for each agent.

Normalization

```
for i=2:length(fieldnames(test.(testcom)))
    agent_allocation_rate{i-1}=test.(testcom).(char(individual(i))).Allocation_rate;
    agent_allocation_time{i-1}=exp(-test.(testcom).(char(individual(i))).Allocation_Time*0.0001);
    agent_satisfaction{i-1}=test.(testcom).(char(individual(i))).Satisfaction;
    agent_provisioning{i-1}=exp(-test.(testcom).(char(individual(i))).Provisioning_Time*0.0001);
    agent_distance{i-1}=test.(testcom).(char(individual(i))).Distance./(length(fieldnames(test.(testcom)))-1);
    agent_latency{i-1}=test.(testcom).(char(individual(i))).Latency./simulation_time_mat{1,1}{1,1}(2);
    agent_usage{i-1}=exp(-test.(testcom).(char(individual(i))).Usage*0.0001);
    agent_messages{i-1}=test.(testcom).(char(individual(i))).Messages./n_mess(i);
end
```

`test.(testcom).(char(individual(i)))` selects the metrics for the agent `individual(i).` In the agent_allocation rate are then saved the values for agent i Allocation rate. Then the normalization for the time metrics is （WP4 D1 pag.31）

```
exp(-time*Beta),
```

where Beta is a parameter of time. This formula is applied also for `Provisioning_Time` and `Usage.`

Finally, Agent Message indicator is normalized taking the ratio with the total message agent number `(n_mess(i)),` and The distance is obtained with the total agent number `(length(fieldnames(test.(testcom)))-1).`

Upper level indicators

```
for i=1:length(fieldnames(test.(testcom)))-1
    m_alloc_time(i)=mean(agent_allocation_time{1,i});
    m_satisfaction(i)=mean(agent_satisfaction{1,i});
    m_provisioning(i)=mean(agent_provisioning{1,i});
    m_distance(i)=mean(agent_distance{1,i});
    m_latency(i)=mean(agent_latency{1,i});
    m_usage(i)=mean(agent_usage{1,i});
    m_messages(i)=mean(agent_messages{1,i});

    std_alloc_time(i)=std(agent_allocation_time{1,i});
    std_satisfaction(i)=std(agent_satisfaction{1,i});
    std_provisioning(i)=std(agent_provisioning{1,i});
    std_distance(i)=std(agent_distance{1,i});
    std_latency(i)=std(agent_latency{1,i});
    std_usage(i)=std(agent_usage{1,i});
    std_messages(i)=std(agent_messages{1,i});

end
```

The above lines evaluate the standard deviation and mean indicators for the main metrics. This script show the indicator for each agent.

```
%% Second layer: mean values over agents

    f_allocation_rate(k)=nanmean(a_l_r);
    f_allocation_time(k)=nanmean(m_alloc_time);
    f_satisfaction(k)=nanmean(m_satisfaction);
    f_provisioning(k)=nanmean(m_provisioning);
    f_distance(k)=nanmean(m_distance);
    f_latency(k)=nanmean(m_latency);
    f_usage(k)=nanmean(m_usage);
    f_messages(k)=nanmean(m_messages);

    f_std_allocation_rate(k)=nanstd(a_l_r);
    f_std_time(k)=nanmean(std_alloc_time);
```

```
f_std_satisfaction(k)=nanmean(std_satisfaction);
f_std_provisioning(k)=nanmean(std_provisioning);
f_std_distance(k)=nanmean(std_distance);
f_std_latency(k)=nanmean(std_latency);
f_std_usage(k)=nanmean(std_usage);
f_std_messages(k)=nanmean(std_messages);
```

This lines compute the final index mean and standard deviation, giving a single indicator for the experiment. It is the value plotted in the spider plots.

Finally the main indicators `ODM, IC, std_IC, std_ODM,` and the final `L`

```
ODM(k)=1-(f_allocation_rate(k)+f_satisfaction(k)+      f_allocation_time(k)+f_provisioning(k))/4;
s_ODM(k)=(f_std_allocation_rate(k)+ f_std_time(k)+f_std_satisfaction(k)+f_std_provisioning(k))/4;
IC(k)=(f_distance(k)+f_messages(k)+f_usage(k))/3;
s_IC(k)=(f_std_distance(k)+f_std_messages(k)+f_std_usage(k))/3;

%% Final index
final(k)=0.5*(ODM(k))+0.5*s_ODM(k)+0.5*IC(k)+0.5*s_IC(k).
```

Annex D – setup of the strategy for the experiment analyzing the effect on message failure on the catallactic strategy

cs.conf:

cs1 bs3
cs2 bs1 bs2
cs3 bs1 bs4

bs.conf

```
bs1 bs1 bronze r1 3 r2 3
bs2 bs2 gold r4 2
bs3 bs3 bronze r1 25 r3 10
bs4 bs4 bronze r4 33 r5 25
```

arb.conf

```
arb1 r1 50 r2 30 r3 30
arb2 r4 50 r5 50
arb3 r1 50 r3 44 r4 45
```

market_decentral.properties

```
############################
# decentral market setup #
############################



##################
# Basic Services #
##################
```

```
################
#bs2
################

bs2.seller.minPrice = 30
bs2.seller.maxPrice = 35
bs2.buyer.minPrice = 30
bs2.buyer.maxPrice = 35
bs2.hard.lower.limit = 15
bs2.hard.upper.limit = 45

bs2.resource.itemids = r4_0


################
#bs3
################

bs3.seller.minPrice = 55
bs3.seller.maxPrice = 65
bs3.buyer.minPrice = 55
bs3.buyer.maxPrice = 65
bs3.hard.lower.limit = 25
bs3.hard.upper.limit = 85

bs3.resource.itemids = r1r3_0


################
#bs1
################

bs1.seller.minPrice = 55
bs1.seller.maxPrice = 65
bs1.buyer.minPrice = 55
bs1.buyer.maxPrice = 65
bs1.hard.lower.limit = 25
bs1.hard.upper.limit = 85

bs1.resource.itemids = r1r2_0


################
#bs4
################
```

```
bs4.seller.minPrice = 55
bs4.seller.maxPrice = 65
bs4.buyer.minPrice = 55
bs4.buyer.maxPrice = 65
bs4.hard.lower.limit = 25
bs4.hard.upper.limit = 85

bs4.resource.itemids = r4r5_0


####################
#     Products     #
####################



####################
#r1r2_0
####################

r1r2_0.seller.minPrice =50.0
r1r2_0.seller.maxPrice =60.0
r1r2_0.buyer.minPrice =50.0
r1r2_0.buyer.maxPrice =60.0
r1r2_0.hard.lower.limit =20.0
r1r2_0.hard.upper.limit =80.0
r1r2_0.baseunit.r1= 1
r1r2_0.baseunit.r2= 1
r1r2_0.resourceids = r1 r2

####################
#r1r3_0
####################

r1r3_0.seller.minPrice =50.0
r1r3_0.seller.maxPrice =60.0
r1r3_0.buyer.minPrice =50.0
r1r3_0.buyer.maxPrice =60.0
r1r3_0.hard.lower.limit =20.0
r1r3_0.hard.upper.limit =80.0
r1r3_0.baseunit.r1= 1
r1r3_0.baseunit.r3= 1
r1r3_0.resourceids = r1 r3

####################
```

#r4_0
####################

r4_0.seller.minPrice =25.0
r4_0.seller.maxPrice =30.0
r4_0.buyer.minPrice =25.0
r4_0.buyer.maxPrice =30.0
r4_0.hard.lower.limit =10.0
r4_0.hard.upper.limit =40.0
r4_0.baseunit.r4= 1
r4_0.resourceids = r4

####################
#r4r5_0
####################

r4r5_0.seller.minPrice =50.0
r4r5_0.seller.maxPrice =60.0
r4r5_0.buyer.minPrice =50.0
r4r5_0.buyer.maxPrice =60.0
r4r5_0.hard.lower.limit =20.0
r4r5_0.hard.upper.limit =80.0
r4r5_0.baseunit.r4= 1
r4r5_0.baseunit.r5= 1
r4r5_0.resourceids = r4 r5

####################
arb item ids
####################

arb.itemids =r1r2_0 r1r3_0 r4_0 r4r5_0

parameters_catnets.conf

```
#
# This file contains all the parameters required for OptorSim
# using the Properties class to store this information.
#
# Aln configuration files

aln.topology.file = examples/300A_500N_W1/topology.conf
aln.bs.file = examples/300A_500N_W1/bs.conf
aln.arb.file = examples/300A_500N_W1/arb.conf
cs.configuration.file = examples/300A_500N_W1/cs.conf


#Number of complex services to be submitted
number.complexservices = 10000
#
# The categories of users available are:
# (1) Simple - wait cs delay between submitting CSs
```

```
# (2) Random - wait uniform random time between 0 and 2 * cs delay
users = 1
#
# The choice of the policy for the ComplexServiceDispatcher is:
# (1) random
# (2) queue length
#
policy = 1
#
# The cs delay is the interval in ms between the
ComplexServiceDispatcher
# submitting each CS.
#
#cs.delay = 600
cs.delay = 1000
#
# The choice of access pattern generators is:
# (1) SequentialAccessGenerator - BSs are accessed in order.
# (2) RandomAccessGenerator - BSs are accessed using a flat random
#        distribution.
# (3) RandomWalkUnitaryAccessGenerator - BSs are accessed using a
#        unitary random walk.
# (4) RandomWalkGaussianAccessGenerator - BSs are accessed using a
#        Gaussian random walk.
# (5) RandomZipfAccessGenerator - BSs are accessed using a
#         Zipf distribution
#
access.pattern.generator = 1
# Shape parameter for Zipf-like distribution > 0
shape = 0.85
#
# The random seed for deciding which cs are chosen can be random or
# fixed.
#
random.seed = no
#
# The maximum queue size is the maximum number of CS the CsHandler
# will keep in its queue.
#
max.queue.size = 2000
#
# The time (in ms) it takes each BS to be executed
#
#bs.execution.time = 800
bs.execution.time = 1000
#
# The choice of market model is:
# (1) Catallactic
# (2) Centralised
market.model = 1
##########################################
# central market parameters              #
##########################################

# Clearing Policy for the Service Market
# (1) Call Market
# (2) Continuous
```

```
market.central.service.clear = 2

# Call Market Clearing Interval for the Service Market
# Defines after how many ms the market will be cleared
market.central.service.clearinterval = 400

# Clearing Policy for the Resource Market
# (1) Call Market
# (2) Continuous
market.central.resource.clear = 1

# Call Market Clearing Interval for the Resource Market
# Defines after how many ms the market will be cleared
market.central.resource.clearinterval = 400

##########################################
# decentralized market parameters        #
##########################################

# market initialisation
market.decentral.file =
examples/300A_500N_W1/market_decentral.properties

# randomize initial price range
price.range.randomize = 0

# min price range
#price.range.min = 5

# connect the prices of the service market and resource market
# if swichted on, the basic service seller's outcome is the budget
# of the basic service buyer on the resource market
# values: yes/no
#markets.connect = no
markets.connect = yes

# resource model selection (resource)
# values: shared, dedicated
resource.model = dedicated

# how to select proposals
# 0 = fifo - one shot (working)
# 1 = fifo - multi-shot (NOT working)
# 2 = best price - one shot (working)
# 3 = best price - multi shot (NOT working)
cfp.selection.model = 0

# maximum number of co-allocated resources
max.coallocation = 0

##############################################
# negotiation stuff                          #
# THIS PERHAPS HAS TO BE MODIFIED FOR CATNETS #
##############################################

# TODO document this parameter
negotiation.flag = yes
```

```
# actually the number of sites contacted
cfp_ann.hop.count = 3


# actually the number of sites contacted
learning.hop.count = 3

# discovery timeout
discovery.timeout = 500

# negotiation timeout
timeout = 5000
timeout.reduction.factor = 0.5

# size of messages (in Kbytes) - size = 0
# implies instantaneous message delivery
message.size = 2
#
```

STRATEGIES:

Strategy 1:
```
###################
# setup learning #
###################

# send plumages
maturityThreshold = 5
#maturityThreshold = 1

# receive plumages
courterThreshold = 20
#courterThreshold = 1
#courterThreshold = 5

# crossover probability
crossoverProbability = 0.20

# mutation probability
#mutationProbability = 0.05
#mutationProbability = 0.25
mutationProbability = 0.7


# ring size
ringSize = 10000

# crossOverSelectionModel
```

```
# (0) select plumages which are better than my plumage
# (1) select best received plumage
#crossOverSelectionModel = 1
crossOverSelectionModel = 0

# init float gene
gaussWidth = 0.1
min = 0.001
max = 0.999

###################
# setup genotype #
###################

# randomize genotype
# values: yes/no
#genotype.randomize = no
genotype.randomize = yes

# if randomize == no, use this genotype
genotype.acquisitiveness = 0.05
genotype.satisfaction = 0.99
genotype.priceStep = 0.5
genotype.priceNext = 0.05
genotype.weightMemory = 0.9
```

Strategy 2:

```
###################
# setup learning #
###################

# send plumages
maturityThreshold = 5
#maturityThreshold = 1

# receive plumages
courterThreshold = 20
#courterThreshold = 1
#courterThreshold = 5

# crossover probability
crossoverProbability = 0.20

# mutation probability
mutationProbability = 0.05
#mutationProbability = 0.25
```

```
#mutationProbability = 0.7


# ring size
ringSize = 10000

# crossOverSelectionModel
# (0) select plumages which are better than my plumage
# (1) select best received plumage
#crossOverSelectionModel = 1
crossOverSelectionModel = 0

# init float gene
gaussWidth = 0.01
min = 0.001
max = 0.999

###################
# setup genotype #
###################

# randomize genotype
# values: yes/no
genotype.randomize = no
#genotype.randomize = yes

# if randomize == no, use this genotype
genotype.acquisitiveness = 0.05
genotype.satisfaction = 0.99
genotype.priceStep = 0.5
genotype.priceNext = 0.05
genotype.weightMemory = 0.9
```

Annex F: Scenario config for the “Second experiment”

Scenario config for Brite:
GUI_GEN.conf

```
#This config file was generated by the GUI.

BriteConfig

BeginModel
        Name =  3               #Router Waxman = 1, AS Waxman = 3
```

```
        N = 2000                #Number of nodes in graph
        HS = 1000               #Size of main plane (number of squares)
        LS = 100                #Size of inner planes (number of squares)
        NodePlacement = 1       #Random = 1, Heavy Tailed = 2
        GrowthType = 1                  #Incremental = 1, All = 2
        m = 2                   #Number of neighboring node each new node connects to.
        alpha = 0.15            #Waxman Parameter
        beta = 0.2              #Waxman Parameter
        BWDist = 2              #Constant = 1, Uniform =2, HeavyTailed = 3, Exponential
=4
        BWMin = 1024.0
        BWMax = 1024.0
EndModel

BeginOutput
        BRITE = 1       #1=output in BRITE format, 0=do not output in BRITE format
        OTTER = 0       #1=Enable visualization in otter, 0=no visualization
EndOutput

GUI_GEN_OPTROSIM.conf
#This config file was generated by the GUI.
BeginOptorSimModel
        ResNum = 3              #Resources Number
        ResMaxQuantity = 100                    #Resource Max Quantity
        ARBNumber = 3                   #Available Resource Bundle Number
        ARBMaxResNum = 3        #Available Resource Bundle Max Number
        BSNumber = 3                    #Basic Service Number
        CSNumber = 3                    #Complex Service Number
        CSMaxCardinality = 3                    #Complex Service Max Cardinality
        FailProbMin = 0.0       #Node Min Failure probability
        FailProbMax = 0.0       #Node Max Failure probability
        QualityNumber = 4               #Quantity Number
        Quality0 = platinum
        Quality1 = gold
        Quality2 = silver
        Quality3 = bronze
EndOptorSimModel

BeginOptorSimModel2
        allocationMechanism = 1                 #0 Centralized; 1 Catallactic
        agentsNum = 2000                #Agents Number
        csSchedule = 0                  #CS Schedule 0= All, 1= random set
        csaPercentage = 20              # Percentage of CSAs
        bsaPercentage = 40              # Percentage of BSAs
        raPercentage = 40               # Percentage of RAs
        bsaDistrProb = 0                # BSA Ditr. of Prob.
```

```
        csaDistrProb = 0                # CSA Ditr. of Prob.
        raDistrProb = 0                 # RA Ditr. of Prob.
EndOptorSimModel2

BeginOptorSimModel_BSTable
        bs1 = 33.0
      bs2 = 33.0
      bs3 = 33.0
EndOptorSimModel_BSTable

BeginOptorSimModel_ARBTable
        arb1 = 33.0
      arb2 = 33.0
      arb3 = 33.0
EndOptorSimModel_ARBTable

CATNETS Simulator configuration

CS.conf
cs1 bs2
cs2 bs1 bs3 bs2
cs3 bs2 bs3

BS.conf
bs1 bs1 bronze r1 20 r3 3
bs2 bs2 bronze r2 52
bs3 bs3 bronze r2 6 r3 1

ARB.conf
arb1 r1 21 r3 10
arb2 r2 25 r3 1
arb3 r2 59


Market_decemtralized.properties

############################
# decentral market setup #
############################


#################
#bs1
#################

bs1.seller.minPrice = 55
```

```
bs1.seller.maxPrice = 65
bs1.buyer.minPrice = 55
bs1.buyer.maxPrice = 65
bs1.hard.lower.limit = 25
bs1.hard.upper.limit = 85

bs1.resource.itemids = r1r3


#################
#bs2
#################

bs2.seller.minPrice = 30
bs2.seller.maxPrice = 35
bs2.buyer.minPrice = 30
bs2.buyer.maxPrice = 35
bs2.hard.lower.limit = 15
bs2.hard.upper.limit = 45

bs2.resource.itemids = r2


#################
#bs3
#################

bs3.seller.minPrice = 55
bs3.seller.maxPrice = 65
bs3.buyer.minPrice = 55
bs3.buyer.maxPrice = 65
bs3.hard.lower.limit = 25
bs3.hard.upper.limit = 85

bs3.resource.itemids = r2r3


####################
#     Products     #
####################



####################
#r2
####################
```

r2.seller.minPrice =25.0
r2.seller.maxPrice =35.0
r2.buyer.minPrice =25.0
r2.buyer.maxPrice =35.0
r2.hard.lower.limit =20.0
r2.hard.upper.limit =40.0
r2.baseunit.r2= 52
r2.resourceids = r2

####################
#r1r3
####################

r1r3.seller.minPrice =50.0
r1r3.seller.maxPrice =60.0
r1r3.buyer.minPrice =50.0
r1r3.buyer.maxPrice =60.0
r1r3.hard.lower.limit =20.0
r1r3.hard.upper.limit =80.0
r1r3.baseunit.r1= 20
r1r3.baseunit.r3= 3
r1r3.resourceids = r1 r3

####################
#r2r3
####################

r2r3.seller.minPrice =50.0
r2r3.seller.maxPrice =60.0
r2r3.buyer.minPrice =50.0
r2r3.buyer.maxPrice =60.0
r2r3.hard.lower.limit =20.0
r2r3.hard.upper.limit =80.0
r2r3.baseunit.r2= 6
r2r3.baseunit.r3= 1
r2r3.resourceids = r2 r3

####################
arb item ids
####################

arb.itemids =r1r3 r2 r2r3

Parameter_catnets.conf

```
#
# This file contains all the parameters required for OptorSim
# using the Properties class to store this information.
#
# Aln configuration files

aln.topology.file = examples/2000A_2000N_1/topology.conf
aln.bs.file = examples/2000A_2000N_1/bs.conf
aln.arb.file = examples/2000A_2000N_1/arb.conf
cs.configuration.file = examples/2000A_2000N_1/cs.conf


#Number of complex services to be submitted
number.complexservices = 100000
#
# The categories of users available are:
# (1) Simple - wait cs delay between submitting CSs
# (2) Random - wait uniform random time between 0 and 2 * cs delay
users = 1
#
# The choice of the policy for the ComplexServiceDispatcher is:
# (1) random
# (2) queue length
#
policy = 1
#
# The cs delay is the interval in ms between the ComplexServiceDispatcher
# submitting each CS.
#
#cs.delay = 600
cs.delay = 1000
#
# The choice of access pattern generators is:
# (1) SequentialAccessGenerator - BSs are accessed in order.
# (2) RandomAccessGenerator - BSs are accessed using a flat random
#       distribution.
# (3) RandomWalkUnitaryAccessGenerator - BSs are accessed using a
#       unitary random walk.
# (4) RandomWalkGaussianAccessGenerator - BSs are accessed using a
#       Gaussian random walk.
# (5) RandomZipfAccessGenerator - BSs are accessed using a
#        Zipf distribution
#
access.pattern.generator = 1
# Shape parameter for Zipf-like distribution > 0
shape = 0.85
```

```
#
# The random seed for deciding which cs are chosen can be random or
# fixed.
#
random.seed = no
#
# The maximum queue size is the maximum number of CS the CsHandler
# will keep in its queue.
#
max.queue.size = 20000
#
# The time (in ms) it takes each BS to be executed
#
#bs.execution.time = 800
bs.execution.time = 1000
#
# The choice of market model is:
# (1) Catallactic
# (2) Centralised
market.model = 1
##############################################
# central market parameters              #
##############################################

# Clearing Policy for the Service Market
# (1) Call Market
# (2) Continuous
market.central.service.clear = 2

# Call Market Clearing Interval for the Service Market
# Defines after how many ms the market will be cleared
market.central.service.clearinterval = 400

# Clearing Policy for the Resource Market
# (1) Call Market
# (2) Continuous
market.central.resource.clear = 1

# Call Market Clearing Interval for the Resource Market
# Defines after how many ms the market will be cleared
market.central.resource.clearinterval = 400

##############################################
# decentralized market parameters        #
##############################################
```

```
# market initialisation
market.decentral.file = examples/2000A_2000N_1/market_decentral.properties

# randomize initial price range
price.range.randomize = 0

# min price range
#price.range.min = 5

# connect the prices of the service market and resource market
# if swichted on, the basic service seller's outcome is the budget
# of the basic service buyer on the resource market
# values: yes/no
#markets.connect = no
markets.connect = yes

# resource model selection (resource)
# values: shared, dedicated
resource.model = dedicated

# how to select proposals
# 0 = fifo - one shot (working)
# 1 = fifo - multi-shot (NOT working)
# 2 = best price - one shot (working)
# 3 = best price - multi shot (NOT working)
cfp.selection.model = 0

# maximum number of co-allocated resources
max.coallocation = 0

################################################
# negotiation stuff                            #
# THIS PERHAPS HAS TO BE MODIFIED FOR CATNETS #
################################################

# TODO document this parameter
negotiation.flag = yes

# actually the number of sites contacted
cfp_ann.hop.count = 3


# actually the number of sites contacted
learning.hop.count = 10

# discovery timeout
```

```
discovery.timeout = 500

# negotiation timeout
timeout = 5000
timeout.reduction.factor = 0.5

# size of messages (in Kbytes) - size = 0
# implies instantaneous message delivery
message.size = 2
#
# Outputs negotiation information to negotiation.log. Can slow the
# simulation down a little bit.
#
negotiation.log = yes
#
##############
# Time Model #
##############
#
# use advanced grid time (yes) or not (no)
#
time.advance = yes
#
#####################################
# Output path for metrics log file #
#####################################
metrics.path = ./log/
#
#
#################################################################
## THE FOLLOWING PARAMETERS COULD BE USELESS FOR CATNETS #
#################################################################
##############################
# BandwidthReader stuff     #
##############################
# flag to switch background traffic on or off
background.bandwidth = no
#
# The directory in which your background bandwidth data files are stored
data.directory = examples/bw_data/edg_testbed/
#
# The datafile to use when no other background data are available.
# For EDG, lyon_to_cern_ave.numbers is good; for UK dl_to_ncl_ave.numbers is good.
default.background = lyon_to_cern_ave.numbers
#
# The time of day used as starting point. Should be in hours, with minutes after
```

```
# the decimal point e.g. 22.5 for 22:30, and must be on the hour or half-hour.
time.of.day = 0.0
#
#############
# GUI stuff #
#############
#
# Options to use the GUI and histogram browser
#
# gui = no
# histogram.browser = no
#
# The file with the map information
#
# map.info = examples/gui/europe.coords
#
##############
# Statistics #
##############
#
# Level of statistics to be printed out at the end of the simulation
# (1) None
# (2) Simple - only stats for the whole grid
# (3) Full - full stats for all elements on all sites
#
# statistics = 3
#
# No background traffic
bandwidth.configuration.file = examples/edg_testbed_bandwidths.conf
#
# automatically multiplied by scale factor
#
# dt = 1000000
#
```

Learning configuration:

```
##################
# setup learning #
##################

# send plumages
maturityThreshold = 5
#maturityThreshold = 1

# receive plumages
```

```
courterThreshold = 5
#courterThreshold = 1
#courterThreshold = 5

# crossover probability
crossoverProbability = 0.20

# mutation probability
#mutationProbability = 0.05
#mutationProbability = 0.25
mutationProbability = 0.7


# ring size
ringSize = 10000

# crossOverSelectionModel
# (0) select plumages which are better than my plumage
# (1) select best received plumage
#crossOverSelectionModel = 1
crossOverSelectionModel = 0

# init float gene
gaussWidth = 0.01
min = 0.001
max = 0.999

####################
# setup genotype #
####################

# randomize genotype
# values: yes/no
#genotype.randomize = no
genotype.randomize = yes

# if randomize == no, use this genotype
genotype.acquisitiveness = 0.05
genotype.satisfaction = 0.99
genotype.priceStep = 0.5
genotype.priceNext = 0.05
genotype.weightMemory = 0.9
```

This report describes the work done and results obtained in third year of the CATNETS project. Experiments carried out with the different configurations of the prototype are reported and simulation results are evaluated with the CATNETS metrics framework. The applicability of the Catallactic approach as market model for service and resource allocation in application layer networks is assessed based on the results and experience gained both from the prototype development and simulations.

ISSN 1864-9300