ECONSTOR
Make Your Publications Visible.

A Service of

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Eymann, Torsten et al.

Working Paper

Preliminary specification and design documentation for software components to achieve catallaxy in computational systems

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik, No. 2

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

Suggested Citation: Eymann, Torsten et al. (2007) : Preliminary specification and design documentation for software components to achieve catallaxy in computational systems, Bayreuther Arbeitspapiere zur Wirtschaftsinformatik, No. 2, Universität Bayreuth, Lehrstuhl für Wirtschaftsinformatik, Bayreuth, https://nbn-resolving.de/urn:nbn:de:bvb:703-opus-3062

This Version is available at:
https://hdl.handle.net/10419/52645

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

WWW.ECONSTOR.EU

Mitglied der Leibniz-Gemeinschaft

Lehrstuhl für
Wirtschaftsinformatik

Information Systems
Management

No. 2

2005

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik

Torsten Eymann / Werner Streitbergern / Michael Reinicke / Felix Freitag / Pablo Chacin / Isaac Chao / Björn Schnizler / Daniel Veit

Preliminary specification and design documentation for software components to achieve Catallaxy in computational systems

Bayreuth Reports on Information Systems Management

UNIVERSITÄT
BAYREUTH

ISSN 1864-9300

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

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

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

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

Authors:

Torsten Eymann, Werner Streitberger, Michael Reinicke, Felix Freitag, Pablo Chacin, Isaac Chao, Björn Schnizler, Daniel Veit

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

D2.1 Preliminary specification and design documentation for software components to achieve Catallaxy in computational systems

Torsten Eymann, Werner Streitberger, Michael Reinicke (Bayreuth)
Felix Freitag, Pablo Chacin, Isaac Chao (Barcelona)
Björn Schnizler, Daniel Veit (Karlsruhe)

Executive Summary.
CATNETS EU IST-FP6-003769 Project Deliverable D2.1
This Report is about the preliminary specifications and design documentation for software components to achieve Catallaxy in computational systems.

Document Id. CATNETS/2005/D2.1/v1.0
Project CATNETS EU IST-FP6-003769
Date Date 2005-02-28
Distribution Public

Copyright © 2005 Issuer

CATNETS Consortium

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

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

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

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

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

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

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

Changes

Version	***Date***	***Author***	***Changes***
1.0	2005-02-15	MRWS	First release candidate draft
1.1	2005-03-08	TE	Added sections on existing taxonomies and Catallaxy background
1.2	2005-03-18	TEMR	Finalized draft

Content

1 Introduction

This deliverable describes the work done in the first 6 months of task 2.1, *" Software components*: Specification and design of software components, based on theoretical and experimental experience from related and the assessment projects, so that Catallaxy can be implemented in any computational system including the simulation and prototype software" from WP2, "Fundamental components*"*. The document is divided in four parts: The introduction with a placement of the CATNETS idea to existing Grid taxonomies, a description of the Catallaxy concept as such, the presentation of the fundamental components identified so far, and the mapping of the concept to the candidate ALN implementations for the "proof-of-concept" prototypes in later work packages.

The findings of task 2.1 come in two parts. This deliverable D2.1 presents the *preliminary* findings so far, which are needed for achieving a common understanding of the Catallaxy and the requirements for both simulation and prototype implementation. Feedback from these joint efforts will enter into a reviewed and more detailed version D2.2, which will present the *final* specification of the fundamental components. D2.2 is scheduled for Month 18 (February 2006).

2 Identification of Fundamental Components in Related ALN

The focus of this section is on the presentation and evaluation of existing mechanisms for allocating resources in a Grid-like Application Layer Network (ALN). ALNs are software architectures that allow the provisioning of services requiring larger amounts of resources, which can be obtained from computing systems connected over simple communication infrastructures such as the Internet. In general, Grid computing, Peer-to-Peer networks, On-Demand Computing and Service-oriented Architectures can be subsumed under this category. A particular resource allocation problem in these concepts is how to match the distributed demand for a service, with an existing, but unclear supply situation.

Using self-organization for such computing system problems, instead of a centralized matchmaker, has recently gained attention by the start of large industrial research concepts like IBM's *Autonomic Computing* or HP's *Adaptive Computing* initiatives. The key motivation aspect for self-organization lies in the increasing size and complexity of today's information systems, which has led to a non-negligible growth of their control costs. Autonomic Computing uses a biological paradigm as a design and control metaphor, the autonomic nervous system (Kephart and Chess). The core properties of the Autonomic Computing concept, the CHOP circle of self-configuring, self-healing, self-organization and self-protection is an electronic realization of the respective mechanisms of the human body.

Abundant biological paradigms distract from the existence of self-organizing resource allocation mechanisms elsewhere, which could, and have been used for engineering and controlling computer systems. In the physical world, for example, the proven ability of a free-market economy to adjudicate and satisfy the conflicting needs of millions of human agents recommends it as a decentralized organizational principle (Wellman; Kephart, Hanson et al.; Eymann and Morito 2004).

Applying Economic concepts to allocating or scheduling resources in computing systems is not a new idea (see (Huberman 1988; Clearwater) for overviews). An early attempt at using economic ideas has been Agoric Open Systems (AOS) (Miller and Drexler; Lavoie, Baetjer et al.). AOS were defined as software systems that use market mechanisms for resource allocation, and encapsulate information, access paths and resources in objects traded by economic actor processes. Similar projects have been Mariposa (Stonebraker, Devine et al.), Popcorn (Regev and Nisan), and Spawn (Waldspurger, Hogg et al.).

The basic problem can be characterized by having a number of processors, supplying computing power to a demand situation composed of computation jobs. The particular question is how supply and demand can be matched to each other, if the actual situation on both sides is unclear. In closed environments, e.g. parallel computing, this question usually can be assumed away, as the number of processors is fixed and the arrival of computational jobs is deterministic.

2.1 Service Provisioning Phases

However, the advent of large, open distributed networks of processors, like in Grid computing, has spurred new interest in this question. Generalizing, to match a particular computation request to a processor service in a Grid, four phases have to be conducted: service discovery, matching requests to services, scheduling the matched services and finally execution (Krauter, Buyya et al. 2001; Nabrzyski, Schopf et al. 2003).

2.1.1 Service Discovery

Finding service instances is a computationally demanding task. Existing sophisticated approaches for service discovery have been realized using flooding algorithms or distributed hash tables (DHT) (Ratnasamy, Francis et al. 2001; Balakrishnan, Kaashoek et al. 2003). The result of the service discovery phase is a list of candidate service provider instances. In this article, we assume that more than one service can provide access rights, and more than one client demands access – otherwise, the discovery phase would be trivial to implement and usual client/server mechanisms apply.

2.1.2 Service Selection / Matching / Scheduling

For all kinds of ALN, different taxonomies have already been published to capture the existing diverse resource allocation approaches in a common framework.

Wolski et al. distinguish between using (1) *centralized omnipotent resource control* or (2) *localized application control* (Wolski, Brevik et al. 2003). This distinction is also made in Figure 1, in the horizontal axis. The first is usually not a scalable solution either in terms of execution efficiency or fault robustness, because the broker is a bottleneck and single point of failure. The second approach can lead to unstable resource assignments as agents adapt to compete for resources.

Most of the actual research relies on the existence of a centralized point of information. GARA (Roy and Sander 2003), LEGION (Grimshaw, Wulf et al. 1994), ECOGRID (Abramson, Buyya et al. 2002), PLANETLAB (Chun, Culler et al. 2003) and RADAR (Rabinovich and Aggarwal) are among the examples for the centralized category. Condor-G (Frey, Tannenbaum et al. 2002), DARWIN (Chandra, Fischer et al. 2001), and most Globus-based implementations (Foster, Kesselman et al. 1999) typically use a centralized matchmaker instance to evaluate the candidate list.

The matchmaker instance selects the apparently optimal match from the list, according to global optimization considerations on latency, distance or bandwidth usage, depending on the current network state. The requesting client receives one singular matching partner. Clients and service providers update the centralized resource broker in a continuous frequency about their requests and effective availability.

Decentralized mechanisms, like in most file sharing networks, e.g. Gnutella (Adar and Huberman 2000) or Kazaa, have no central point to collect supply and demand before matching. Each client decides for himself which service provider to match to based on technical parameters like estimated download time. APPLES (Casanova, Obertelli et al. 2000), WINNER (Arndt, Freisleben et al. 1999) or MARS (Gehring and Reinefeld 1996) are other examples for localized control. The problem of localized control is the missing assurance on allocation stability (Wolski, Brevik et al. 2003).

Two formal approaches to the Grid resource allocation problem are control theory (Burghes and Graham 1980) and economics. As a means of achieving stability without relying on a centralized information base, Wolski *et al.* propose to use Economic mechanisms in favour of Control theory, whose findings they describe as "elusive". The distinction between "Eco-

nomic" approaches and others corresponds to the second category in Figure 1, on the vertical axis.

MILLENIUM (Chun and Culler 2000), G-COMMERCE (Wolski, Brevik et al. 2003), and ECOGRID (Buyya 2002), Nimrod/G (Buyya, Abramson et al. 2002) are examples given for those *computational economies*.

Centralized approaches implement auctioneers, like in ECOGRID, or comparable electronic marketplace instances (Gomoluch and Schroeder 2003), which collect bids and offers from the Grid nodes, and match supply and demand like a stock market mechanism does. As an example, Wolski *et al.* have in G-COMMERCE, *agents* (producers and consumers), the commodity objects (tagged with a price), and a centralized institution they call "The First Bank of G". This institution uses a "tâtonnement" (Walras 1954; Cheng and Wellman 1998) sequential auctioneering approach, combined with a polynomial method for finding general market equilibria by Smale (Smale 1976). The price setting of the individual producers and consumers uses local knowledge about the resources, and the single-variable utility functions are expressed in budget units only.

Subcategories of computational economies are *commodities markets* and *auction markets*. In a commodity market, resources are interchangeable, and a buyer accessing a resource does so from a pool of equivalent choices, without the ability to specify which resource exactly will be purchased. In an auction market, buyers explicitly specify the particular resource or good instance to access. The fine distinction is that in a commodity market the (single) price is set, so that all buyers and sellers are satisfied, while the auction market has specific prices, for each buyer-seller pair. In the centralized control case, this price setting is done by the (commodity) resource broker or the auctioneer, while in the localized case, each seller is its own auctioneer, or sets the price according to some local optimization rule.

Summarizing, either the client (decentralized case) or a resource broker (centralized case) have to select a match out of several possible pairings, which satisfies both parties. Satisfaction can be ideally measured either by technical parameters (fast execution time, low bandwidth usage, minimal communication overhead) or by translating these to economic metrics, e.g. utility as minimal direct access costs or as a function of waiting time saved. In principle, existing service matching mechanisms can thus be visualized as a 2x2 matrix shown in Figure 1.

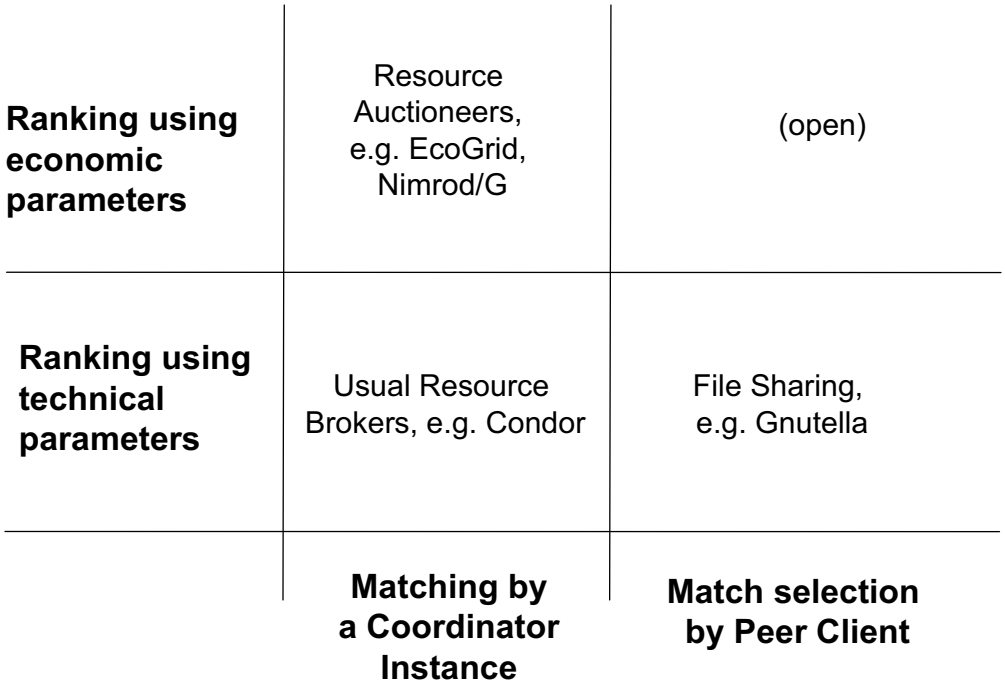


Figure 1: A portfolio of Grid Service matching mechanisms

In another taxonomy, Gomoluch and Schroeder (Gomoluch and Schroeder 2003) distinguish resource allocation concepts according to the dimensions:

1. *State-based vs Model-based:* Are the allocations based on a current snapshot of the system state (*state-based*), which is expensive to obtain, or on a model, which predicts the system state and which may be inaccurate (*model-based* or *predictive*)?

2. *Pre-emptive vs Non-pre-emptive:* Are tasks assigned to hosts once (*non-pre-emptive*) and then stay there, or can they migrate if it turns out at a later stage that it is advantageous to leave the machine (*pre-emptive*)?

Again, these two questions unfold a 2x2 portfolio, where most of the existing Grid allocation models fit into.

State-based, non-preemptive strategies are the easiest to implement. A known environment state is evaluated, and a new state of increased optimality is computed (usually by a central institution). The execution phase allocates demand statically to supply, until the next computation cycle occurs. SPAWN and POPCORN are examples for this category.

Pre-emptive strategies allow the migration of an ongoing job from one resource to the other. Technically, this creates a lot of security and stability problems; in the context of the initial matching process, the distinction is not necessary and shall not be followed here.

Model-based approaches to resource allocation involve two very challenging problems: how to obtain an initial model/prediction and how to adapt the model as time passes. In principle, the participants predict a future state of the environment, and use a model-based strategy to

act accordingly, in order to improve their outcome. In the centralized case, the global outcome will be considered by the central institution (assumed that it is not possible, though, to sense the current environment state). In the decentralized case, all participants try in parallel to improve their own outcome, with a priori unknown effects on the total. A model-based example is Nimrod/G (Buyya, Abramson et al. 2002).

Krauter *et al.* (Krauter, Buyya et al. 2001) have developed a comprehensive multi-dimensional taxonomy for Grid resource management systems for distributed computing, taking into account "machine organization within the Grid, resource model, dissemination protocols, namespace organization, data store organization, resource discovery, QoS support, scheduler organization, scheduling policy, state estimation, and the rescheduling". Some of these categories will be highlighted.

The machine organization category distinguishes between flat vs. hierarchical organization. In a flat organization, all machines directly communicate with each other without going through an intermediary. This corresponds to localized application control, if the machines are considered to be peers. Hierarchical communication has one or more machines being superior to others; in the centralized control case, the resource broker would be considered superior. Implicitly, communicating flat vs. hierarchical corresponds also to the question, whose optimization goal is superior. Flat organizations with peers have all local goals equal; in hierarchical organizations, being "above" usually means having priority if goals conflict.

In the resource discovery and dissemination category, the terms are defined as "discovery is initiated by a network application to find suitable resources within the Grid. Dissemination is initiated by a resource trying to find a suitable application that can utilize it. […] Resource dissemination is categorized by the approach taken to updating the resource information." The authors distinguish between a batch/periodic dissemination approach, where information about the resource is push-sent in time intervals to the prospective clients or a broker instance, or pulled by the latter. In comparison, Online/On-Demand dissemination has push-sending by the resources, whenever a change in resource supply occurs or an actual access request is received.

Different combinations of scheduler organization, state estimation, rescheduling, and scheduling policy classifications can be implemented in a Grid resource management system, as those categories are orthogonal to each other. Of those, the scheduler organization category distinguishes classically between centralized, hierarchical and decentralized approaches, which needs not be repeated. Of more interest is the question of how the current environment state is estimated.

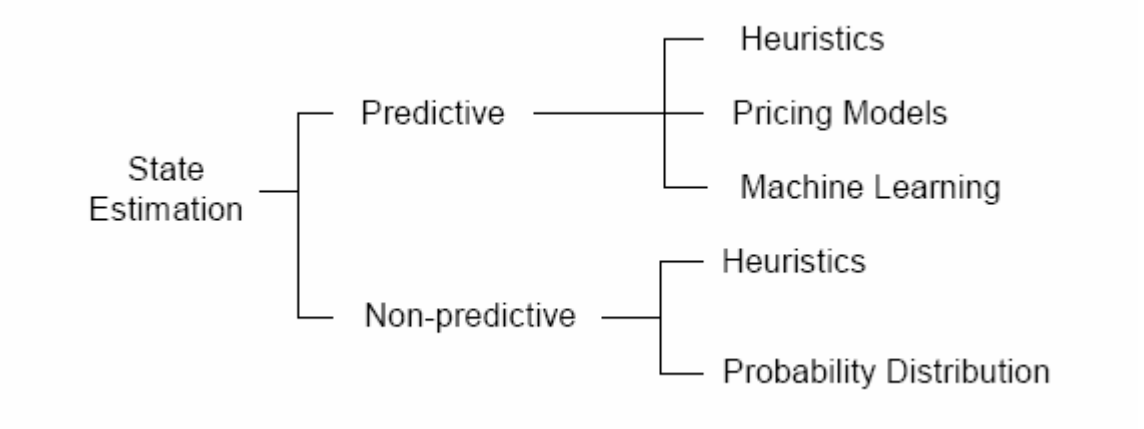


Figure 2: State Estimation Taxonomy (Krauter, Buyya et al. 2001)

"In Grid systems, state estimation is always done on partial or stale information due to information propagation delay in large distributed systems" (Krauter, Buyya et al. 2001). Differing

from (Gomoluch and Schroeder 2003), the authors thus focus on model-based approaches, where they only distinguish whether the models take only historical data into account (non-predictive), or try to predict future states. Of particular interest for CATNETS are those approaches, which include adaptive and pricing prediction. Pricing prediction uses market dynamics and price signals, to estimate future states. Adaptive, machine-learning approaches try to optimize behaviour in multi-dimensional, potentially unknown distributions of several variables. Both approaches can complement each other, which CATNETS will exploit.

Summarizing all taxonomies, the CATNETS scheduling approach is defined by *localized application control* using *economic mechanisms*, as the single agents all set their prices and bids locally. However, it has aspects both of a commodity market (the single agents trying to adapt to a fictive market price) as well as an auction market (the agents bidding for a specific resource, even in the Catallaxy case). CATNETS employs client-based economic decision-making mechanisms with a *model-based prediction* of the system state (Gomoluch and Schroeder 2003) and allocation via *bargaining models* (Buyya, Abramson et al. 2002). Within the same layer (resources vs. application), it uses a *flat communication* model. Dissemination of information is *on-demand*, as the price signals reflect the actual (but constantly changing) situation at the time of requesting access. The prices also allow the participants to *predict* future states of the Grid; improving the accuracy of that prediction is subject of *machine-learning* adaptation.

These characteristics form the upper right corner of Figure 1, which is only sparsely populated otherwise. One apparent implementation has been MojoNation (Mojo Nation 2003), but it failed due to missing control of pricing schemes and control of money supply.

2.1.3 Execution

For interfacing with the execution phase, the matching phase ends regardless of the mechanism with the matched partners agreeing on quality of service (QoS) parameters (e.g. scheduled time and duration, guaranteed lower bounds on bandwidth, storage or processor time), compiled and fixed in the service level agreement (SLA).

In the execution phase, client access the scheduled services, usually according to the information stated in the SLA. If the SLA gets breached by either side, contingency procedures have to be taken. While posing serious questions and motivating further research, these security issues are not in the focus of the current project.

3 The Catallaxy as an economic self-organization framework

The purpose of this section is to provide an insight into the economic fundament of the decentralized Grid scheduling algorithm. As there is no common, agreed-upon understanding on the actual implementation of the Market mechanism, and it is at the current time impossible to excerpt a formal, mathematical description of that mechanism from Economics literature, the following text will concentrate on the concept. However, as the final purpose is an actual Grid implementation, we will try to get as specific as possible.

Friedrich August von Hayek (Hayek, Bartley et al.), and other Austrian economists understood the market as a decentralized coordination mechanism, as opposed to a centralized command economy (note that Austrian, or Neo-Austrian economics, describe an Economics line of thought, rather than only the geographical heritage of its prosponents). Apart from political macroeconomic thoughts, his work also provides concrete insight on the working mechanisms of economic coordination. The emergence of software agent technology and increasing size of information systems leads to the possibility of implementing Hayek's Catallaxy concept and using the ensuing "spontaneous order" as a concrete proposal for both the design and coordination of information systems. However, a formal description of this self-organizing market mechanism does not so far exist.

The Catallaxy concept bases on the explicit assumption of self-interested actions of the participants, who try to maximize their own utility and choose their actions under incomplete information and bounded rationality (Simon). The term Catallaxy comes from the Greek word "katallatein", which means, "to barter" and at the same time, "to join a community." The goal of Catallaxy is to arrive at a state of coordinated actions, the "spontaneous order", which comes into existence through the bartering and communicating of the Community members with each other and thus, achieving a community goal that no single user has planned for (Hayek, Bartley et al. 1989). The main characteristics of the Catallaxy (Hoppmann) are that

- Participants work in their own interest to gain income. Every system element is a utility maximizing entity, which requires the definition of utility itself, of means to measure and compare income and utility, and to express a desire to reach a defined goal. For humans, these definitions have not necessarily to be explicit or thoroughly defined; for information system elements, this explicitness is required.
- Human action always takes place in a world of uncertainty (Horwitz 2003). Participants subjectively weigh and choose preferred alternatives in order to reach an income or utility maximization goal. In Neo-classical economic theory, the "homo oeconomicus" is a completely rational utility maximizer. He can choose an alternative action out of total knowledge about the environment. Hayek's claim was that such an "objective" choice is not possible because of "constitutional ignorance", that it is (inevitably) impossible to know each and every detail of the environment state. For large and very dynamic information systems, this is inherently true, and overcoming it by central means requires synchronization and restriction of possible actions of the single elements.
- Participants communicate using commonly accessible markets, where they barter about access to resources held by other participants. The development of prices for a specific good, whether they are increasing or decreasing, leads buyers to look for alternative sources of procurement and thus enhances the dynamics of the market. Note that a market here is nothing more than a communication bus – it is not a central entity of its own, which collects all information and matches market participants using some optimization mechanisms, which would contradict "constitutional ignorance".

In human economic systems, these institutions are implicit; for a realization in distributed information systems, the properties of utility maximization, strategies and the exchange of

offers need to be explicitly specified and implemented. Formal descriptions for using economic mechanisms in distributed computing systems can be found in (Ferguson, Nikolaou et al. 1996).

As a blueprint for other possible forms of Service Grids (Gomoluch and Schroeder 2003) or Application Layer Networks, we describe the concept using a simple web services scenario of a PDF conversion service (Catnet Project; T-Online AG):

Adobe's PDF file format is a common exchange file type for mixed text and graphics documents, mostly due to its preservation of layout specifics. The files are created using the (usually locally installed) Acrobat Distiller service, which converts from e.g. Microsoft Word or Postscript files. In an "on-demand" Service Grid, Distiller web services are available in the network, hosted by independent vendors and directly accessible from the software application, competing with each other for the clients' demand. The word-processor client programs transparently address such a networked PDF conversion service instance in the background, without disturbing the user's course of work. Clients and service provider instances bargain on access prices on a case-by-case basis, taking into account the current and prospective development of supply and demand to increase the monetary utility of their respective owners. Services instances are situated on host computers, which, for simplicity, are assumed to provide processor power and storage on a fixed cost basis.

Economic actors are straightforwardly implemented as intelligent software agents (Wooldridge). Agents are embedded in an *environment*; whose state they experience through *sensors*; which lead to a comparison of an actual environment state with a desired environment state using an *internal world model*; and where they try to influence the environment state using *effectors* towards that more desirable state.

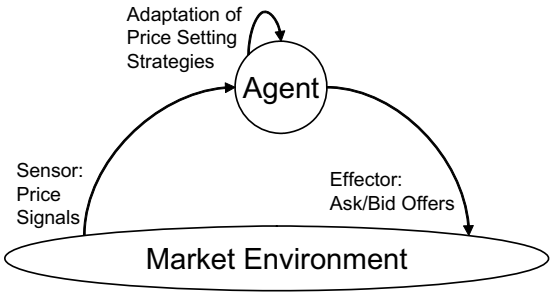


Figure 3: Properties of Digital Business Agents (cf. (Wooldridge))

Figure 3 shows a *Digital Business Agent* (Eymann) working in a market environment. Sensors and effectors are realized as price signals incoming from and outgoing to the market environment. If the agents' utility goals are not met by the present ownership situation, they negotiate with each other in order to maximize utility by exchanging resource access rights (e.g. using an alternating offers protocol (Rosenschein and Zlotkin)). Bartering forms a sequence of effectors and sensors, this leads under partial and bounded knowledge to an adaptation of the agent's internal model. Implementing Edgeworth bartering (Varian), the agents trade bilaterally and secretly with each other, *if* the internal world model prognoses utility increase out of the potential transaction. Setting the price right is the most important action decision. Sellers intent to obtain the highest possible price for the service access they offer, buyers want to pay the lowest possible price for the service. To that respect, the seller offers (*Asks*) will be higher than the reservation prices, while buyer offers (*Bids*) will probably be lower. A too high price

in the face of competition will not lead to many transactions, while a price too low leads to less income per transaction.

The constant price signaling between entities propagates changes in the scarcity of resources throughout the system, and leads to constant adaptation of the system as a whole. Imperfect knowledge makes it thus necessary to adapt the agent's price setting strategies dynamically in order to maximize individual utility. This is achievable using feedback learning algorithms (Evolutionary algorithms, Numerical optimization, or hybrid methods like Brenner's VID model (Brenner), which are all principally interchangeable (Müller and Eymann)).

In our example, three types of market agents appear (see Figure 4): the client agents, the service instance agents and the resource agents (as embodiments of the hardware/network provider). The market environment itself is not a solid object – it is a communication platform, implicitly realized by the network provider, communicating the effector actions of all other agents in the environment.

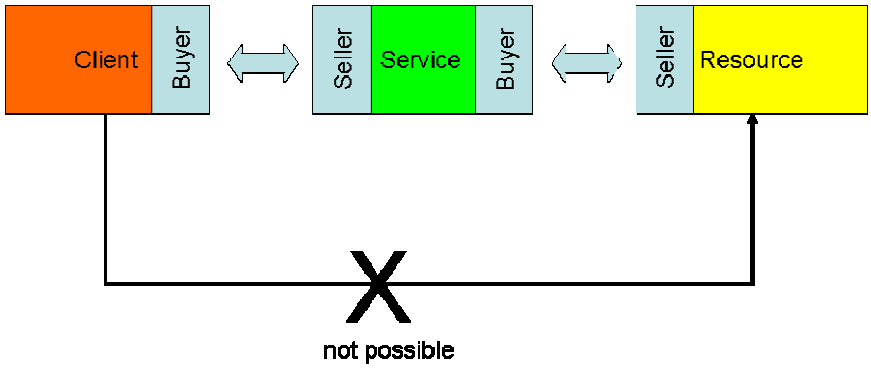


Figure 4: Interaction relations in the Service Market

- The *Client* is a computer program on a certain host, which needs access to a web service to fulfil its design objectives. It tries to access that service at an arbitrary location within the computer network, use it for a defined period, and then continues with its own program sequence. Client programs run on a connected network resource, usually a personal computer. The business strategy of the client computes the fixed cost of purchasing a local copy of Adobe Distiller against the variable cost of using such available On-Demand services, multiplying the forecast number of uses with the access price. If the client user does not need to convert to PDF so often, it will rely on a certain price and availability level of the service provider and refrain from buying an expensive local copy; if the usage frequency is above a certain number, it is in total cheaper to buy that exclusive copy. In an environment where services have to be paid for access, the utility gain of clients is the difference between their *private value* (of what the access is worth) and the actually paid *transaction price*:

$$u_i^C = v_{i,p} - p \tag{1}$$

- A *Service* is an instantiation of a general application function, embodied in a computer program. A *Service Copy* is an instance of the service; a resource computer, which provides both storage space and bandwidth for the access of the service, hosts it. The business model of the service provider leads to set the access price so that the majority of clients decide to rely on the on-demand option. In theory, the price will be equal to the marginal cost of processing the penultimate access, which means that it is exactly so high that the consumer is undecided whether to buy the local copy or access the remote service (pro-

vided that both prognoses the same number of accesses in a given time span). If the service provider is able to distribute several instances (service copies) in the network, he might be able to sell each copy access for a different price, according to the time of day, the geography of the network or the willingness of clients to pay. Each redundant web Service Copy is thus a miniature business, like a retailer's branch store. Like the clients, the service providers also have a private value for service access (*sa*) . In addition, there is private value for buying network resource access (*ra*) from the hosting node:

$$u_j^{SC} = \left(p^{sa} - v_{p,j}^{sa} \right) + \left(v_{p,j}^{ra} - p^{ra} \right) \tag{2}$$

- A *Resource* denotes a host computer, which provides a limited number of storage space and access bandwidth for service transmission. The network connections between the resources are simulated to be of equal length and thus of equal transmission time and costs. The resources and network owner (the network provider) allows service providers and clients to communicate using cables, routers, gateways and other, hardware or software network layer instances. For the usage of these resources, he gains income from all participants – the more participants, the more money can the network provider make. However, more participants means more traffic in the network, and above some level the traffic can get so extensive that the existing resources are no longer sufficient. However, if the dimensioning of resources is too large, the income from the participants might not be high enough that the resource investment is economically justified. In the long run, the network provider will provide enough network resources for the average use, but will be vulnerable to usage spikes. These resources incur costs, and the network provider aims to fill these costs and to make profits by increasing the usage of the resources:

$$u_k^R = p \ - v_{p,k} \tag{3}$$

Summing up all utility functions over the number of respective participants, the parameter Social welfare utility (SWF) measures how the aggregate of all the individual utility is maximized. The equation thus can be written as

$$U_{SWF} = \sum u_i^C + \sum u_j^{SC} + \sum u_k^R \tag{4}$$

After each successful trade, the sum of all utilities of all participants increases. A fictive final state would have maximum overall utility and is Pareto-optimal, which means that no single agent can propose a change that does not decrease any other's utility. However, as ALN nodes appear and disappear dynamically, such a solid state may never be reached. Under the restriction of an imperfect knowledge situation, a total optimal value of SWF can only be measured in hindsight.

4 Fundamental components in CATNETS

Application Layer Networks (ALN) encompass heterogeneous resources by a high number of geographically distributed devices and administrative domains, which are logically coupled together for providing processes on application level. This comprises both computational and data services.

We expect ALNs to be shaped by lots of basic services that can be dynamically combined to value-added complex services (like in Service-oriented architectures (Singh and Huhns 2005)). These basic services require a set of resources, which need to be co-allocated to provide the necessary computing power (like in computational Grids). The orchestration and customization of these basic services and resources can be understood as an inherent service, that must be accomplished by the network as well, due to the complexity and the expertise requirements which must be hidden from the application.

We thus divide the playing field in two layers, the application layer and the resource layer. Layers, both in software (Bachmann, Bass et al. 2004) and networks (Tanenbaum 1996), allow to hide complexity and to provide common interfaces at the vertical touching points.

In these two layers, we contemplate three different roles, which are:

- complex services (application layer),
- basic services (application layer and resource layer) and
- resources (resource layer).

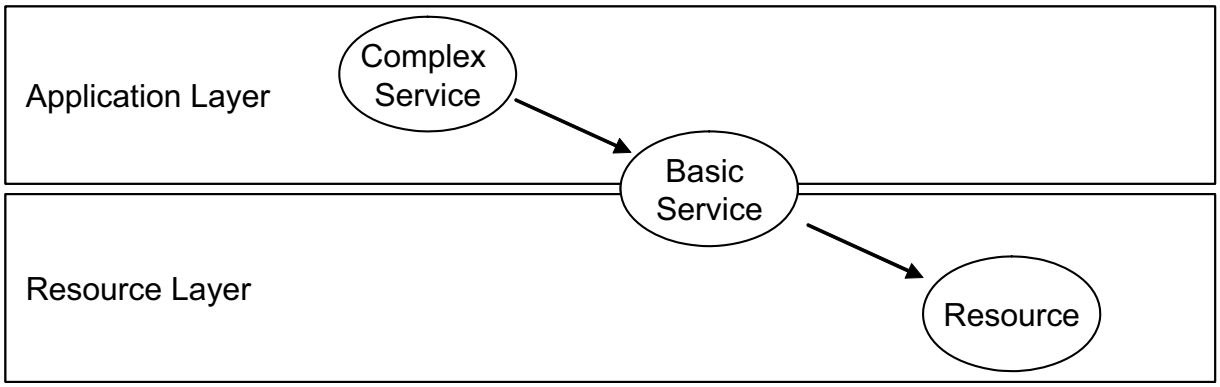


Figure 5: Layered-view on ALNs

Basic services thus offer the interface to accessing computational resources for complex services, while hiding the orchestration and implementation details. In existing Grids, this more user-centered view is not needed, as the computation requests are already expressed in appropriate semantics (in what we call the resource layer). In Service-oriented computing, users define complex business job requirements, which are then broken up into collections or sequences of basic services that together provide the desired functionality.

In both layers, the participants similarly have various objectives, tasks, strategies and demand patterns, which might change dynamically and unpredicted during life time. Traditional approaches, using centralized policies require complete state information which is not available in dynamic and complex networks (Krauter, Buyya et al. 2001). As an acceptable system-wide performance matrix is impossible to define, we use an economics-based paradigm for the management or resource allocation and orchestration (Buyya 2002).

The economics-based paradigm is derived from human economies, where decentralisation and heterogeneity are successfully managed. These models, when involving decentralized decision-making, are based on exchanging and acting on price signals. The participants work for their own utility; thus, they evaluate the signals received and act according to some utility function, which predicts an utility increase out of the effect of that action. On a system-wide

scale, a bird's eye view on such market dynamics shows continuous matching of demand and supply, a case of emergent coordination. If trying to deliberately construct such automated markets, we are searching for coordination mechanisms, which take the dynamicity of the market into account.

This section will provide a detailed presentation of the market models, which are understood as fundamental components for the Catallaxy Realisation. Two different market approaches are compared, the baseline and the Catallactic market. The baseline market is a centralized market with one centralized matchmaker. This instance tries to fulfil all clients' requests by matching providers' offers with clients' requests (see D1.1). A completely decentralized market is the Catallactic market. In this market there is no central matchmaker, the clients negotiate with the services directly, abstaining from global knowledge.

Both markets are two stage markets. One market is contemplated as a service market, a client (complex service) requests a set of basic services to achieve the desired functionality. The other market is a resource market. On this market the basic service buys and co-allocates the required resources, which allow it to deliver its own part of the required service functionality.

For a circumstantial comprehension of the markets, the terms used are definable as:

Complex Service: (Former client in CatNet.v1) A modular software application which needs a set of basic service capabilities for fulfilling its goals.

Complex Service Logic: Translates the requirements of a complex service to a set or sequence of modular basic services.

Basic Service: A module includable in a complex service.

Basic Service Logic: Translates basic service (depending on the multi-attributive requirements of the clients) in resource policy.

Co-Allocator: Tries to accomplish resource policy, obtained from Basic Service Logic. Policy can be split to several Local Resource Managers.

Local Resource Manager (equals manageable entity): Manageable interface to Resources, which hides resources hardware (low level) details.

Resource: Low level resources can be obtained by Local Resource Managers.

In the remainder of this section, we compare our market model with existing implementations in our own assessment project CATNET, Bittorrent, PlanetLab, and Coral. The purpose is to nearer specify what functionality and what components need to be placed in the particular participating objects.

4.1 Market Model

Current Grid Computing architectures exhibit fairly static resource infrastructure which is connected by physical stable links (e.g. enterprise grid). The shift to a pervasive grid, that could exist ubiquitously, demands for a more dynamic consideration of resources and connec-

tions. In CatNet.v1 the used market model addressed grids with a comparably static resource infrastructure. An overview of the market is shown in Figure 6. The *Client* requests a service transaction and inquires the available *service copies* for an offer. These copies will after reception of the request try to contract the *resources* on their current resource node to be able to offer the service. If *resources* are available and contractable, the *service copies* begin to negotiate with the *client*. This is an m:n:1 relationship. The *client* and the *service copy* negotiate in an m:n market, as services are available for everybody and clients could contact all *service copies*. *Service copies* could fail. Possible reasons for failure can be software updates or loss of connectivity.

Service Copy and *Resource* build an n:1 relation, as only *local resources* can be contacted. This is contemplated as the second market. *Clients* request demands in random time intervals, set by the simulation environment. The *service copy* could be understood as business process to be included in application software on the *client* side.

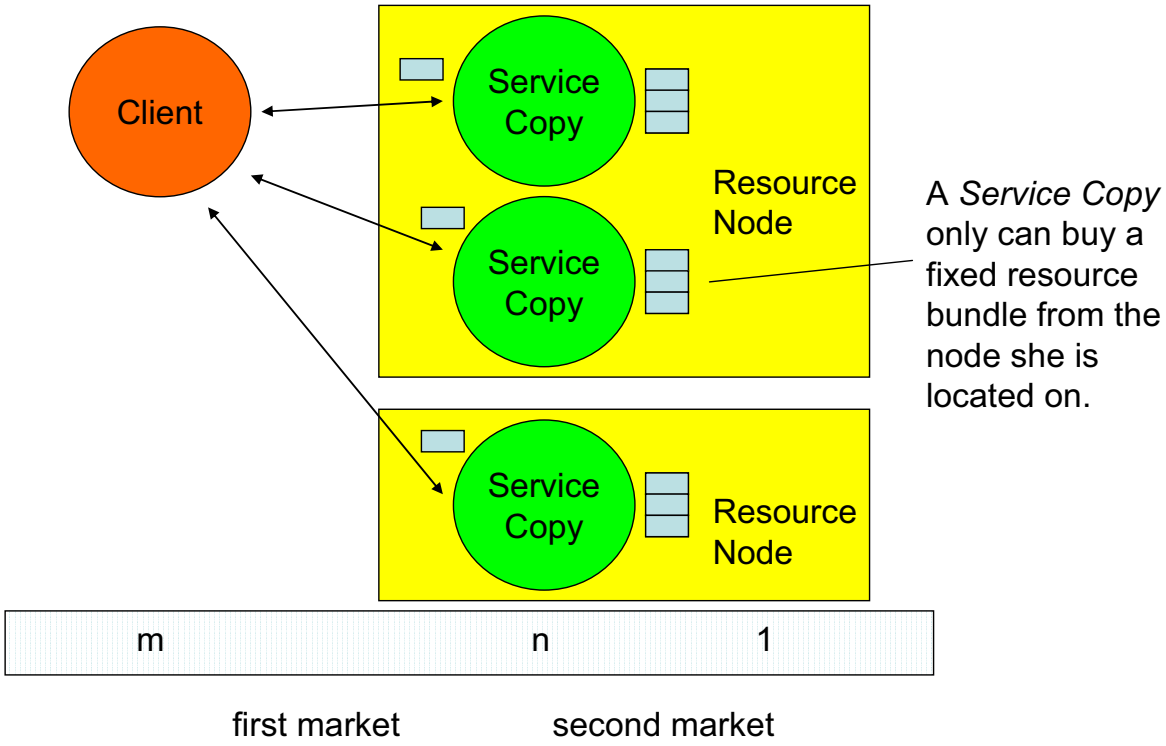


Figure 6. Market model of predecessor CatNet.v1

CATNETS extends the CatNet.v1 market to a complete two stage market, to match dynamicity of application areas in future grid technologies (Figure 7). The market is understood as a decentralized control mechanism for services and resources. To satisfy the needs of future application domains in grid and peer-to-peer networks, several alterations to the primary model are proposed:

The *client* is replaced by a complex service, and could be represented by a modular software application which needs (remote) basic service capabilities for execution. The intention behind this perception is that the client and the complex service are shielded from the details on the resource layer and site. Ideally, both only interact with the basic services which are located in the system. Resources are not visible for them at any time. The complex service contains a service selector instance that selects and contracts the basic service.

The *basic service* is split in the *basic service logic* and a *resource allocator*. The logic is able to negotiate with the complex service and to translate the requirements for service execution in a resource specification (e.g. CPU, storage and quality of service requirements, etc.).

The *resource allocator* gets the resource specification and broadcasts the respective demand to the *local resource managers*. This comprises bundles and co-allocative negotiations. Bundles are understood as an n-tupel of resource types (e.g. CPU, storage, and bandwidth); co-allocation describes obtaining resources for one single service transaction from various *local resource managers* at the same time. Local resource/job scheduling is not in the focus of the project and will not be further analyzed: The *local resource manager* hides all details of the allocation.

On the first market, *complex service* and *basic service* negotiate, whereas *complex service* acts as a buyer, the *basic service* as a seller agent. The same market roles can be found at the resource layer, the resource allocator is the buyer agent, the *local resource manager* acts as seller agent.

Contemplating the second market, it is extended to a *n* to *k* market: *n* service copies can bargain with *k* resource services to fulfil their service demands. This takes dynamic resources into account. Resources are in our view entities that can fail like basic services which are subject to maintenance and inspection procedures or link failures.

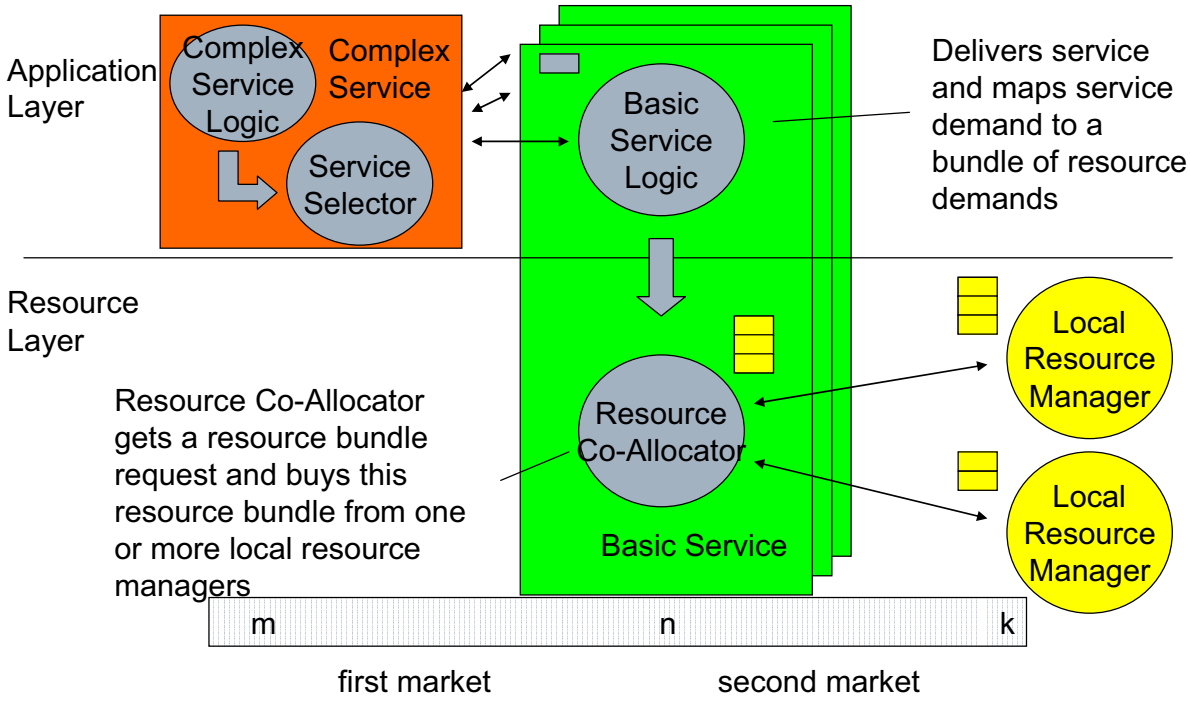


Figure 7. CATNETS market model

In CATNETS, a deviating scenario is possible. In prevailing grid applications, the application layer is not considered in particular; both markets are located on resource layer. The translation to resource specifications is done on the complex service logic that maps its own resource demands to a resource selector, which buys resources from different resource allocation instances. Figure 8 shows this model as an alternative to the generic CATNETS market model.

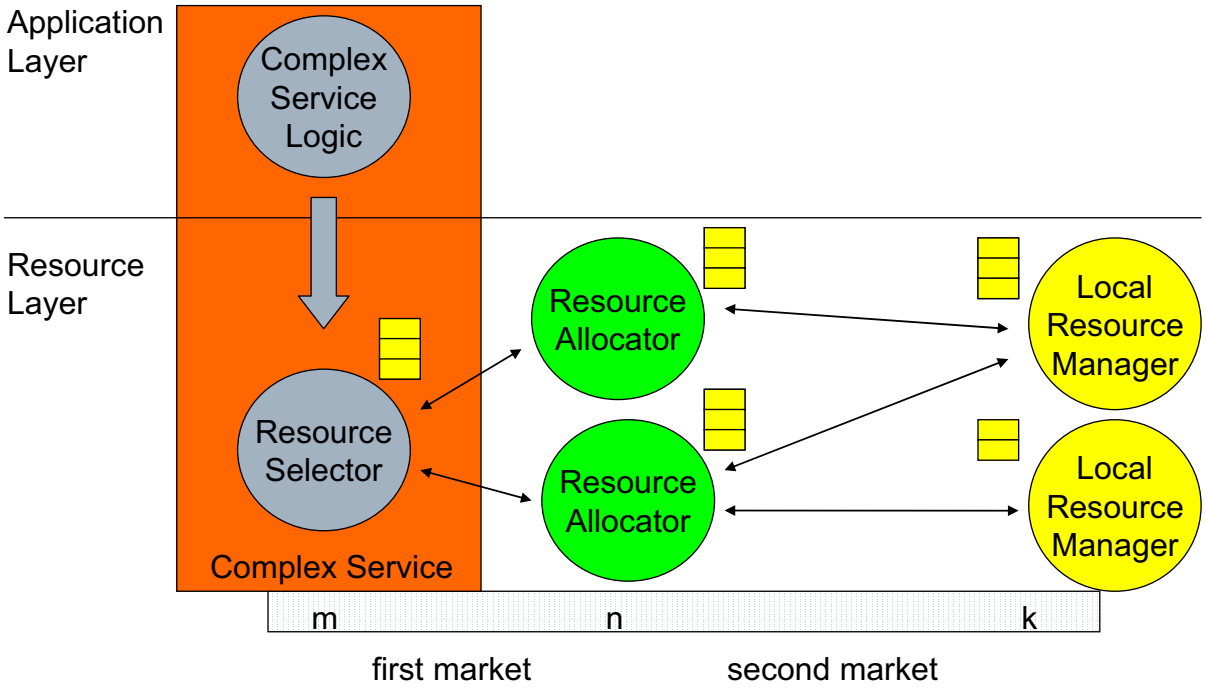


Figure 8. Alternative CATNETS market model

This model includes certain drawbacks concerning complexity on the client side. It is presumed, that the client/complex services owns knowledge about the translation process form his demands to resources. In the upper model the knowledge is split and sourced out to the network of basic services, which allows reducing complexity and performance claims from clients, which gives the opportunity to model light-weighted clients (like mobile phones, PDAs).

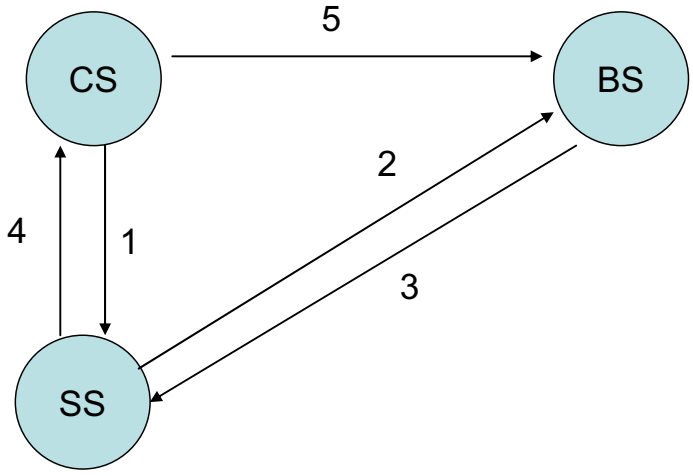


Figure 9. Execution of a service transaction as a high-level sequence diagram

Figure 9 depicts the general sequence of how to deploy a service. The *complex service* (CS) sends a *requestBasicService* to the *Service Selector* (SS), which contacts all discovered and available *basic service* instances in an asynchronous communication (step 1). The *Basic Service* initiates the negotiation process (steps 2 and 3) if it can contract a resource bundle for service completion. The *Service Selector* informs the *complex service* on which service can used for execution of his job (step 4). In step 5 the CS uses the BS according to the services delivered by the SS.

Summing up, CatNet.v1 and CATNETS can be compared like presented in Table 1.

Table 1. Comparison of CatNet.v1 and CATNETS market models

CatNet.v1	CATNETS
• N : M : 1 market	• N : M : K market
• A service can only use the resources he is hosted on.	• A service can use different resources simultaneously
• 3 market roles	• 4 market roles (services contain two market roles)
• Static bundling on the resource market; client cannot use more than one service for fulfilment of his request	• Service-defined bundling on resource market; clients can access several services at the same time
• Sequential negotiations	• Parallel negotiations
• Only service copies may fail	• Services and resources may fail

4.2 Components

Within the market model every participant has a distinct role. Figure 10 depicts the relevant use cases of these roles. All roles will be described in the following section.

- The CS offers itself to the client (*offerComplexService*) which is outside of our focus, so it does not exist in the diagram.

- For job completion the CS requests a BS (*requestBasicService*). The request is generated by the *complex service logi*c which knows which types of *basic services* are necessary for execution (*selectBasicService*). This SS is exclusively activated when there is a request received and is similar to a bill explosion in logistics.

- *SelectBasicService* is called by the *Service Selector*. The *Service Selector* successively calls a search algorithm (*searchService*), a *rankItemList* procedure, a bilateral negotiation procedure (*negotiateItem*) and after each successful transaction a clearing method (*doClearing*). The search should deliver all available basic services which could provide the desired service (*searchService*). This list is thereafter sorted by the ranking process (*rankItemList*). The service selector will then initiate the negotiation with the listed services in the ranked order and his expected service-dependant utility increase (*negotiateItem*).

- The basic service logic, building the opponent in the first market, offers a basic service (*offerBasicService*) and uses the same negotiation procedure like the service selector. Note, that there is no advertising possible, thus this is a demand driven market.

For offering a basic service, the basic service logic needs to contract the required resources (on resource layer). This translation request can be done in advance, during negotiation and after the contract with the complex service.

1. Contracting resources in advance requires a calculated forecast for the future demand (Buyya 2002). For a centralized allocation mechanism this might be suitable as demand and supply fluctuations can be absorbed over the whole network. In decentralized decision-makers, this is quite a complex task, as demand and supply can change rapidly, and the decision-makers will not be able to anticipate this situation by their local knowledge. Therefore, they will be exposed to a higher risk of bankruptcy.

2. Contracting resources after closing the service contract might lead to insufficient resource offers on the resource market and thus to unaccomplishable contracts. Economically considered, this leads to high risks in reputation on the market.

3. Contracting the resources during negotiation is considered as best suited. Before giving a first proposal to the opponent, the negotiation is delayed and a contract is trying to be accomplished with several local resource managers. This has the inherent advantage, that supply changes in the resource market can be transferred immediately to the service market. This reduces risks for the basic service and balances both markets. If there are not sufficient resources available, the basic service needs to contract resources for a higher price and claims that surplus in the ongoing negotiation from the complex service.

We will focus on possibility 3, as it seems to be beneficial.

- For service execution the basic service logic requests a resource bundle (*requestResourceBundle*). The further process of contracting/allocating the resource is done by the resource co-allocator. The selection of a resource bundle is done analogous to the selection of a service, with the exception, that a bundle is requested (*selectResourceBundle*), whereas on service market only one service can be negotiated per request.

- The local resource managers offer resource bundles (*offerResource*). The resource bundle could be a tuple consisting of bandwith, CPU, and storage. The manager is the seller agent of the resource market, having the ability to negotiate with the resource allocator (*negotiateItem*). The negotiation is also initiated by the resource co-allocator.

- Finally, the clearing is invoked by the seller or buyer side of the markets (*doClearing*). The decision on this fact is unimportant, due to the fact that the modelled agents are benevolent.

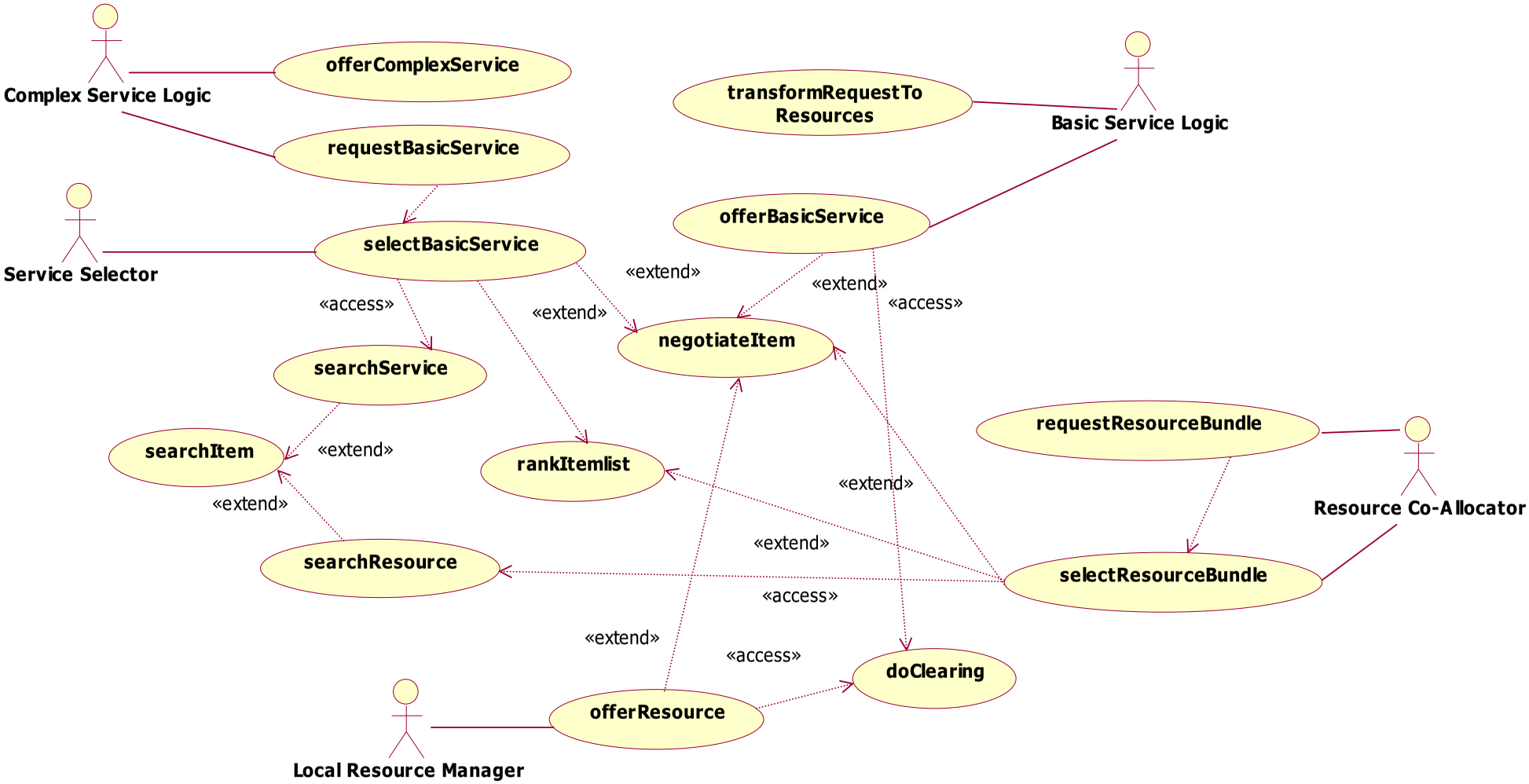


Figure 10. Use case diagram of the service acquisition process

Every player in the market can be modelled as a software agent. The players get their basic functionality (communication, learning algorithms, strategies etc.) from an agent source abstract class. In Figure 11 the service agents contains the buyer and the seller. The specialization and concrete implementation of the agent source is done in the derived components' classes.

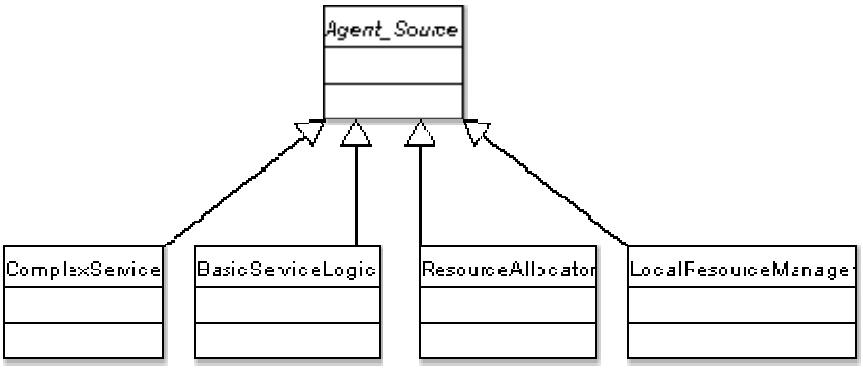


Figure 11. Agent source and specialized agents in CATNETS

The basic service and the resource allocator are modelled as separate agents, as the complexity should be reduced. Both will need a special interface to exchange their transformation specification (prices, market development, etc.) from service supply to resource demand.

4.3 Lifecycle of Agents

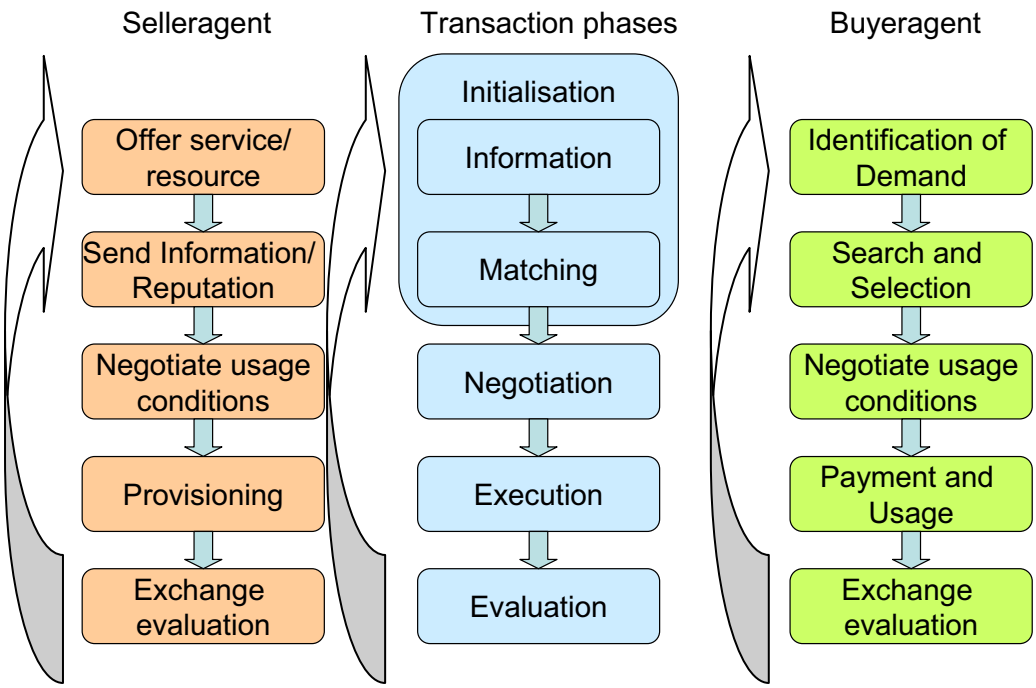


Figure 12. Lifecycle of agents

This section presents a general model of transaction phases and the mappings of this model to the buyer and seller agents. The lifecycle starts with an initialisation phase, which is split into an information subphase and a matching subphase.

- The initialisation phase prepares the agents for the proceeding negotiation phase. In the information phase the buyer agents (resource co-allocator and complex service) try to identify the demand. The seller side (basic service and local resource manager) offers his selling items.
- The matching brings together the buyer and seller agent. The buyer agent initiates this process after specification of his demands. Seller agents can support the buyer agents with additional information of the service or their reputation.
- A parallel, bilateral exchange of information between buyer and seller agents shapes the negotiation phase. The usage conditions between seller and buyer agents are multi-attributive items (like basic service on the first market and resource bundles on the second market).
- The execution phase contains deployment and clearing of the contracted service, which the seller agent delivers on demand. Often the evaluation phase is omitted and the process begins again.
- In CATNETS, the evaluation phase analyzes the business relationship between the buyer and seller. Information about evaluation is exchanged between them. This information is used to optimize the next business relation ship between a seller and buyer agent.

4.4 Interactions between market participants

The sections above gave a short overview over the interaction of the market participants. Here we will present a deeper insight of these interactions and classify them into the widely accepted taxonomies of grid economics literature.
There are several generic models to shape the negotiation behaviour of the agents. A general framework is presented in (Buyya 2002).

- Commodity markets,
- Posted Price model,
- Bargaining model,
- tender/contract-net model,
- auction model,
- bit-based proportional resource sharing,
- Community/coalition/bartering/share holders model and
- monopoly/oligopoly etc.

In our decentralized architecture an iterative bilateral negotiation protocol, similar to a contract-net, is used because we have a state of incomplete information. Both agents approximate to the trade-off point in iterative steps exchanging offers and counter-offers. This process is described as monotonic concession protocol (Rosenschein and Zlotkin 1994).

A preliminary model of our protocol is shown in Figure 13. For clarity, the basic service is split in its buyer and seller side – basic service login (seller side) and resource co-allocator (buyer side).

The complex service requests a basic service, according to his process demand. The basic service translates this request to resource layer and the resource co-allocator and starts the negotiation with several *local resource managers*. The *local resource manager* analyses the request and creates an offer and this process iterates until an agreement (accept) or reject is

reached. If an accept is reached, the resource allocator confirms and informs the basic service about the contracted resources. The basic service continues the negotiation with the complex service, using the information from contracting the resources. The negotiation on resource layer is processed only once. It is impossible to renegotiate a resource contract. The resuming negotiation with the complex service uses the same negotiation protocol and after an accept the payment process is initiated which pays the basic service and the resource. A reject on the service market will lead to a reject ton the resource market.

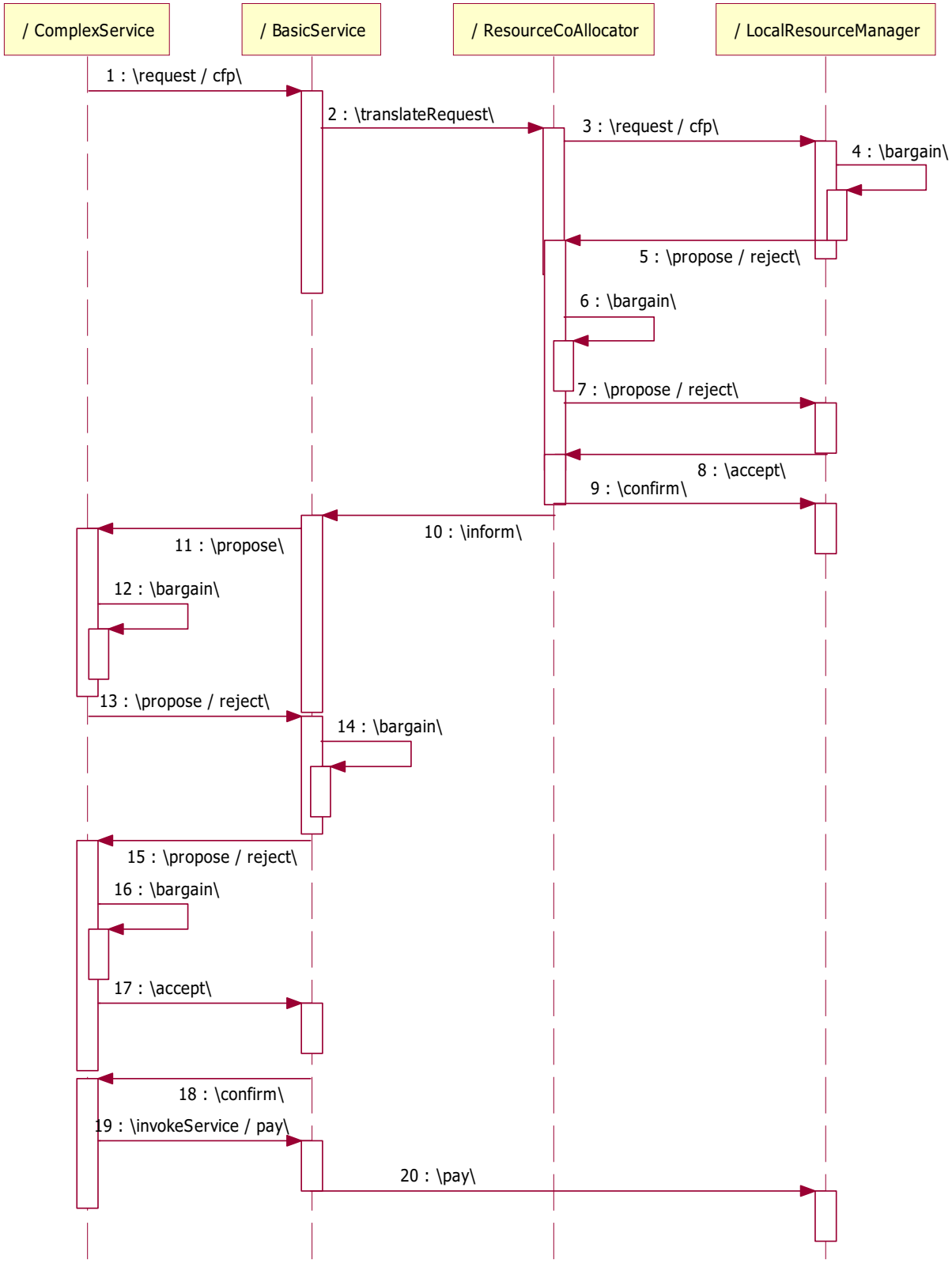


Figure 13. UML sequence diagram of the negotiation between the market participants

4.5 Additional conditions for Catallaxy realisation

For the realisation of the Catallaxy paradigm several additional conditions have to be implemented in the decentralized CATNETS architecture.

The preceding service discovery phase can be discussed and implemented separately, depending on the application concerns. Discovery is necessarily required to initiate the matching/negotiation procedure and therefore must deliver sufficient service offers to allow a selection between those items. However, it does not necessarily involve

Considering the preparation and calculation of price proposals, a negotiation module is required that constitutes the interface between internal perception of the environment and the surrounding (sensor and effector). These negotiation strategies need to use learning mechanisms, to react to changes in the environment and to implement a method that adapts to the behaviour of the surrounding agents.

4.5.1 Service discovery

The main project interest is the service selection process. However, a successful search, delivering an (unsorted) list of suitable services, is a necessary precondition for the sorting and selection process. This is implied by the effect, that clients lack global, persistent knowledge about the system's states in a dynamic surrounding and whether a suitable service/resource exists and is still alive/available for consumption.

Implementing a central catalogue for service discovery, like shown in common web search machines (e.g. Google) or former file sharing systems (Napster etc.) is contemplated as an essential obstacle for a decentralized operation of dynamic application networks and counterproductive for the evaluation, as this implies misallocations resulting form the discovery process. Thus, solely decentralized discovery mechanisms are accounted for evaluation.

4.5.1.1 Unstructured discovery

The simplest decentralized search method is using an unstructured flooding mechanism (Gnutella 2000, Karl Aberer, Magdalena Punceva, Manfred Hauswirth and Roman Schmidt, Improving Data Access in P2P Systems, Matei Ripeanu (Ripeanu 2001), Peer-to-Peer Architecture Case Study: Gnutella Network). Flooding works under the assumption of a nodes' neighbour relations. Queries are not transmitted to a central catalogue, but instead distributed among the peers: A search request is forwarded to all neighbours and all neighbours behave respectively (Figure 14). This flooding is limited by a time-to-live parameter (TTL) that restricts an infinite search. Every node, storing the required item, send a list of all content matching the query to the originating node the same way back and waits for a direct download request by him. Gnutella uses this unstructured overlay network in that the topology of the overlay network and the placement of the resources (files) is largely unconstrained.

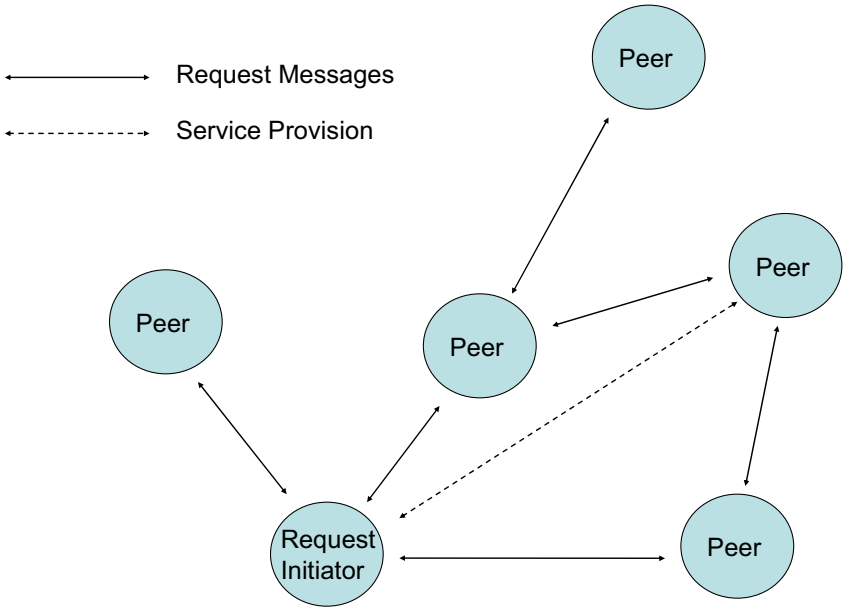


Figure 14. Decentralized Service Discovery in Gnutella

Obviously, the load of each node grows linearly with the total number of queries, which grows with system size. So, this approach is not scalable. It implies that elevated network traffic has to be accepted under the constraint, that an existing service is not reliably discovered in large diameter networks, due to the time-to-live parameter and the decreased search space. Several improvements have been made to increase the search behaviour of the flooding mechanism, and delimit the network consumption of the algorithm. The improvements are mainly developed to reduce the message emergence, like e.g. k-random walker search, adaptive TTL, one hop replication etc.

For example, the GIA algorithm combines some innovations to P2P systems:

- A dynamic topology adaptation protocol, that aligns nodes in short reach of high capacity nodes,
- one hop replication of data pointers, so all neighbours maintain pointers to the content offered by their immediate neighbours,
- an active flow control, to abandon overloaded hot-spots,
- a search protocol, that is based on biased random walks, directing queries to towards high capacity nodes.

Simulations showed that these improvements provide a three to five orders of magnitude in the total capacity of the system and retains robustness to failures. A distinguished melioration of the flooding mechanism is the percolation search algorithm described in (Sarshar, Boykin et al. 2004), which limits the search messages by concentrating on highly connected nodes and building request-individual internal hierarchies in the (power-law) service network. The percolation search algorithm consists of

(i) Content Caching: An initial replication of a node's content list or directory in the random walk visited nodes,
(ii) Query Implantation: A query first executes a short random walk and implants its query request on the visited nodes and
(iii) Bond Percolation: A probabilistic broadcast scheme for propagating queries.

Percolation simulation results showed, that unstructured P2P networks can be made scalable.

Following Gnutella's lead, other decentralized (file sharing) systems have become popular. KaZaA is using FastTrack, which adopts supernodes that have higher bandwidth and connec-

tivity. Associated supernodes contain pointers to the content of the peers and each search request is forwarded from the peers to the supernodes. Only between the supernodes the flooding mechanism is implemented. This approach appears to offer better scaling performance; however, its scalability in comparison to Gnutella has neither been analyzed nor measured (Making Gnutella-like P2P Systems Scalable).

4.5.1.2 Structured discovery

In contrast to unstructured overlay networks, structured search algorithms promise a guaranteed item discovery and a reduced message emergence. The usage of distributed hash tables (DHTs) in CHORD (Balakrishnan, Kaashoek et al. 2003), CAN (Ratnasamy, Francis et al. 2001), Pastry (Rowstron and Druschel 2001), TAPESTRY (Zhao, Kubiatowicz et al. 2001), VICEROY (Malkhi, Naor et al. 2002) offers a guaranteed search and distribute the search process to the connected nodes in the network. The search does not rely on random search behaviour in the network but calculates the closest known node on the straight way to the requested service instance. Thus, a performance of O(log N) can be guaranteed, which improves Gnutella's performance of O(n) (Chawathe, Ratnasamy et al. 2003).

Distributed Hash Tables (DHTs) are a class of systems that provide "hash-table like semantics at Internet scale". The original rationale was to provide a scalable replacement for not scalable Gnutella-like file sharing systems. In recent years a lot of research has been done on these DHTs. All of these proposals are structured overlay networks, where both data placement and overlay topology are completely controlled.

For unstructured networks, churn does not cause big problems as long as it is not disconnected by all of its neighbours. A peer can then re-run the bootstrap algorithm, to replace itself to reconnect to the system and a new location. DHTs, in contrast, are highly vulnerable to churn, that causes significant overhead: Most DHTs require O(log n) repair operations after each failure. Unpredicted failures, where peers cannot inform the network about that expected breakdown, require even more time to discover the failure and replicate lost data and pointers. Anticipating a high churn rate, the overhead caused by the repair operations of the DHTs can easily overwhelm low-bandwidth nodes (Chawathe, Ratnasamy et al. 2003).

Existing networks show complex network characteristics and tolerance to node deletions (Ripeanu, Foster et al. 2002; Saroiu, Gummadi et al. 2002). However, client-based protocols that guarantee the global emergence of scale-free networks with tuneable properties have not been implemented.

4.5.1.3 Evaluation

For the selection of a search algorithm, CATNETS related criteria have to be taken in special consideration. Scalability, simulation ability and applicability for dynamic networks have to be taken into account as both simulation and prototype should behave the same in service discovery to exclusively measure the effect on service selection. However, DHTs lack scalability in dynamic networks, as state changes (e.g. churns) lead to high overhead and might influence the simulation behaviour considerably. They are expected to burden the execution process of the simulation in an unacceptable and unpredictable manner and thus cannot be favoured for implementation in CATNETS, as they constitute a risk for the project goals. Contrariwise, newer, revised flooding algorithms (GIA, Percolation Search) in unstructured overlay networks are under development and offer a scalable and easy implementable mechanism, abdicating the performance losing risks of DHTs.

Thus, we regard the implementation of a simple flooding algorithm as to be best suitable for CATNETS that – in a real application – could be extended to one of the revised flooding mechanisms presented above.

4.5.2 Negotiation

As a basic principle, the negotiation strategy constitutes a search process in a space of potential agreements. The dimension of this search space is identical with the number of negotiation attributes. Thus, a negotiation comprising quality of service, delivery time, and price spans a 3-dimensional search space. In several cases, it is possible to map various attributes into one criteria "price", for example when delivery time affects the buyer's usage and therefore justifies a change of the price. Multi-attributive negotiations are deepened in (Bichler 2001) and D1.1. The following section gives an overview in the negotiation concerns.

4.5.2.1 Type of negotiation

An automatic negotiation in an electronic market is shaped by an interaction of two ore more software agents, exchanging communication acts. These negotiations can be accomplished in two diverse types (Pruitt 1981; Jennings, Faratin et al. 2001), which differ in the handling of the negotiation dimensions:

In *integrative* negotiations, participants exchange information about objectives and priorities to seek for a common solution. This concept is recommendable if the opponents have to accept the negotiation dimensions which cannot be represented by prices. This postulates a cooperation of the opponents for reaching the agreed target.
D*istributive* negotiations imply a participant's step-by-step accept of concessions, bringing both opponents closer in their expectations every negotiation round. Distributive negotiations are marked by existence of a common utility space (Pruitt 1981), that can be represented by a price. Thus, distributive negotiations give the option to reduce the negotiation dimensions. This should result in a null-sum game, the utility one looses can be gained by the opponents and the global utility in the systems remains constant.

4.5.2.2 Goal

The goal is a system wide pareto-optimum that can be consulted as an *acceptable doctrine of general goodness* (Rosenschein and Zlotkin 1994): A solution X is pareto-optimal, if no agent can further meliorate the achieved result without discriminating an opponent. That implies that if solution X is not pareto-optimal, both agents could negotiate a deviating solution that promises pareto-optimality. Sandholm (Sandholm 1996) extends this approach by introducing various additional criteria for the optimality determination: from game theory he uses the Nash-equilibrium that emerges if no agent has an incentive to diverge from his chosen selection.
Translated to prices, this means that pareto-optimality is a state in which no agent can increase his budget without decreasing the budget of other agents (cp. Null-sum game). Utility can be understood as budget increase per transaction and per period, sales volume or other values taken from business economics.

4.5.2.3 Strategy

The definition of a strategy how to reach the objectives of a negotiation is essential for modelling a market. In principle, the initial situation can be described like depicted in Figure 15. A (human) principal defines an indifference price that equals his estimation about the value of the good. For a buyer, this is a maximum price, for the seller a minimum price. So, the utility gain equals the amount between price of the purchase and the indifference price. The start price represents the price where the strategy begins to negotiate. By agreeing concessions, the opponents come closer to the middle and a possible contract. A transaction is unlikely, if the closure zone is empty, which might result when indifference prices do not build an overlapping zone.

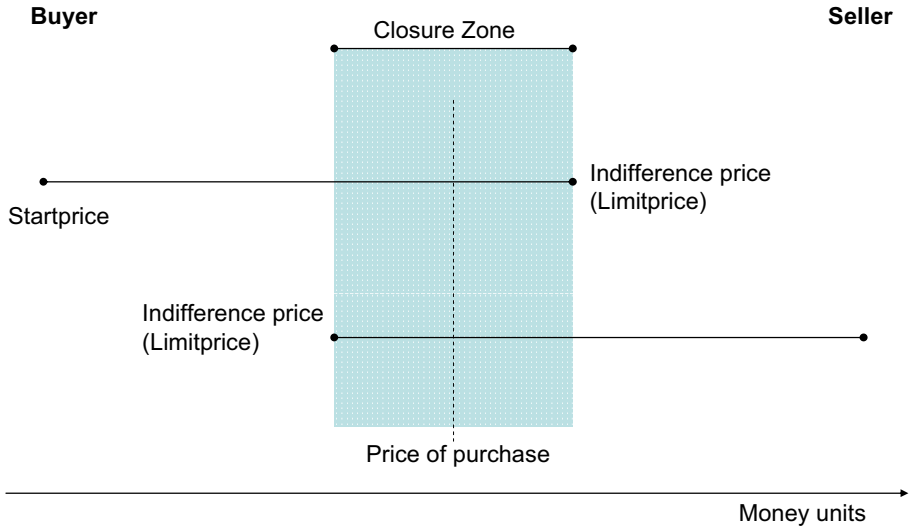


Figure 15. Bilateral negotiation process (De Paula 2000)

The bargaining protocol schematized in Figure 15 can be implemented in different modes:

1) Buyers and sellers give their start prices without agreeing concessions. Thus, a contract can exclusively be accomplished, when the start price of one participant is already in the closure zone. An example is the usage of catalogues, where offer prices are fixed.
2) Only the seller performs concessions and the buyer remains at its start price. This is represented by the Dutch auction.
3) Only the buyer performs concessions and the seller remains at its start price. This is represented by the English auction.
4) Both agents get closer each negotiation step. This monotone sequence of concessions describes a double auction (Friedman 1993).

However, the communication sequence is not sufficient for the description of the negotiation protocol: mostly, several prospects are imaginable when receiving a proposal by an opponent: The agent could reject the proposal, accept it or send a counter-offer. This decision should include past and forecast experiences. The number of alternatives describes the complexity of the decision process; the more alternatives exist, the bigger the search space, which increases the quest for a suitable solution. Though, this may not exceed the time frame that is accepted by the opponent for the reply message/counter proposal. According to (Preist 1998; Lomuscio, Wooldridge et al. 2000), four mechanisms/strategies for negotiation can be differentiated:

1) Rule-based mechanisms are subject to the premise, that all possible states are completely known and the environment remains static during processing. This might be suitable for a market with fixed catalogues, as re-consulting a catalogue with static prices and articles will not change the decision rules. Double auctions, however, cannot be matched, as implementing all decision alternatives when changing prices or articles' attributes is impossible. The conceived market model of CATNETS will inherently comprise dynamic and complex states of the environment; therefore rule-based mechanisms are not appropriate.

2) Argumentative mechanisms include not only the proposal for prices but also purchase-supporting arguments which extend the negotiation dimensions. The arguments highly depend on the application, thus no standards can be set and this implicitly leads to a higher complexity in decision and modelling. Furthermore, it exhibits a deviation of the single dimension negotiations.

3) Game-theoretic approaches assume the market situation to be a multi-stage game between buyer and seller. The strategy results from the analysis of the negotiation problem. For the formulation of a result, the availability of an offer of the counterpart is not necessary; the analysis can rely on prospects. The internal model of the agent comprises a calculus which explicitly includes all probable behaviours of the opponents.

   Multi Agent Systems can be implemented using Game-theoretic approaches, but due to the high computation prerequisites, these systems may be limited to a number of 20 agents (Müller and Eymann 2003). Thus, Game-theoretic approaches are not suitable for the concerned market model, due to the scalability concerns.

4) Heuristic-adaptive approaches assume incomplete knowledge a priori, and therefore they also expect defective decisions. Agents adapt their strategy by relating the behaviour of the market and their own activities (Sathi, Fox et al. 1989; Sycara, Gasser et al. 1989; Cliff and Bruten 1998; Bussmann and Schild 2000). The counteroffer of the opponent is contemplated as feedback of the former own proposal. As the whole spectrum of opponents' proposals cannot be anticipated completely beforehand, the strategy uses forms of "trial and error" to formulate offers. An easy example of use is shown in (Preist 1998) where heuristic rules are combined with easy learning rules. Each agent is willing to negotiate a price that is below (buyer) respectively above (seller) its own price limit. His autonomous decision is to determine a price for selling/buying. The market mechanism is a round-based continuous double auction, and all agents use the same implementation. The implemented heuristics determine the target price of the agents in their negotiation steps. A number of numerical or boolean parameters determine the strategy; these variables will be adapted during the lifetime of an agent. In MAS learning methods like neural networks (Fausett 1994), Q-Learning (Sandholm and Crites 1995), classifier systems (Holland 1992), Bayesian networks (Neal 1996) and evolutionary algorithms (Goldberg 1993) can be identified.
   This implementation is typical for realisation of heuristic-adaptive strategies. From the mentioned mechanisms, heuristic adaptive strategies show the most scalable behaviour (Eymann 2003) and are favoured for adoption in large MAS. Due to the fact, that they show scalability and a good behaviour they will be chosen for implementation in CATNETS.

The following section introduces the depicted learning algorithms and evaluates them for adoption in CATNETS.

4.5.3 Adaptation of strategy

The combination of artificial intelligence and machine learning has become increasingly complex in the last years, adopting evolutionary algorithms, fuzzy logic or neural networks.

Brenner (Brenner 2002) classifies learning methods in non-conscious learning, routine-based learning and belief learning. However, non-conscious learning processes are discussed not to appear in experiments, belief learning focuses on the individual and not on the whole population. For global optimization, belief learning seems not to be suitable.

For bilateral negotiation processes, optimizing global behaviour of a population, routine based learning strategies are most applicable.

Routine based learning strategies describe the learning process on population level. It is acceptable, to model the learning process not granularly on the individual level, as the emerging behaviour of the complete system is in focus of CATNETS.

There are mainly two possible ways how to implement a reinforcement learning strategy: the Roth-Erev model and evolutionary algorithms.

- The Roth-Erev (Erev and Roth 1998) model is a quite simple model, which is adequate for a small, fixed set of actions and strategies. Because of being limited to a small, pre-defined set of actions, the Roth-Erev model is not applicable of CATNETS.

- Evolutionary algorithms are able to deal with a very large set of actions and strategies and allow the sets of strategies increase endogenously. Therefore this type of algorithms is more applicable to CATNETS.

Nevertheless, both models should not be considered to be more than quite crude approximations on a population level of real conscious learning processes. In economic simulations lots of research efforts on evolutionary algorithms can be found.

We selected the STDEA (Smith Taylor Decentralized Evolutionary Algorithm) for CATNETS (Smith and Taylor 1998). This algorithm showed good results in the predecessor project CatNet. Other possible strategies are numerical optimization procedures. These algorithms search for the parameter configuration which makes the maximum profit. Eymann an Müller (Müller and Eymann 2003) could show, that results in similar learning mechanisms showed very similar results.

The STDEA is a decentralized evolutionary algorithm, which means that it has no global evaluation metric (fitness value), which is used in Genetic Algorithms (Goldberg 1993) to separate the underperforming participants. A fundamental quality of the mechanism is the decentralized communication and fitness evaluation, using locally 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 object is evaluated, avoiding premature unification of the genotype. Sender and recipient

remain anonymous. If a certain maturity threshold of received plumages 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 behaviour of the population. An implementation of the STDEA algorithm is shown in Figure 16.

When comparing with other, numerical optimization strategies (Press, Teukolsky et al. 2002) and decentralized learning, it must be admitted, that e.g. Powell's algorithm as well as the simplex method provide better results than the STDEA, because evolutionary algorithms perform a routine based learning, which constitutes a slower learning process than observed in reality. However, the substantial advantage of optimization strategies to decentralized learning mechanisms becomes obvious if the size of the population is varied: In the case of one single agent, the numeric algorithms take advantage of their directed search, in contrast to the random exploration of the decentralized learning mechanisms. Increasing the size of the population the optimizer will reach its performance limits, whereas the learning mechanisms do not lack scalability and even perform better with an increasing number of agents.

A mixed model is OVID (Optimized Variation-Imitation-Decision) (Brenner 1996; Müller and Eymann 2003). The OVID model presents an option of combining the advantages of the genetic algorithm STDEA, the numeric optimization procedure of the simplex method and the imitating and directed exploring behaviour of human cognitive processes. This algorithm does not depend on a constant information flow between the agents, but can meaningfully optimize its own behaviour at any time. It is thus more robust than STDEA and yields better results than pure numerical optimization approaches. The OVID model is currently only evaluated in a test bed. There are no results using this algorithm in real application. Therefore, it is considered to be a risk for the project using this model for learning.

In CATNETS the use of the STDEA algorithm is recommended, which has proved to be able to handle an elevated number of agents in simulation and prototype.

Figure 16 presents the UML Class diagram of the learning algorithm. The abstract class *Gene* describes general methods which are similar in all concrete data types.

The class *BooleanGene* is derived from Gene and implements the (boolean) mutation process, allowing exclusively changes from 1 to 0 or 0 to 1. *FloatGene* implements a creep mutation, where the genes are changed in small steps.

The *Genotype* consists of the properties *acquisitiveness, delta_change, delta_jump, satisfaction, weightmemory,* and *reputation.* These values are constant for each agent and can exclusively be adapted by learning mechanisms.

Plumage defines the object which is used for exchange of genotype and fitness information and therewith describes the learning information for the other participants.

IMutable and *IRandomizeable* standardize the interfaces, which can be called by the agents' individual learning mechanisms.

Smith and Taylor (Smith and Taylor 1998) have set the configuration values defining the frequency of the exchange of learning information, adding received plumages to the stack and selecting received plumages for crossover and mutation.

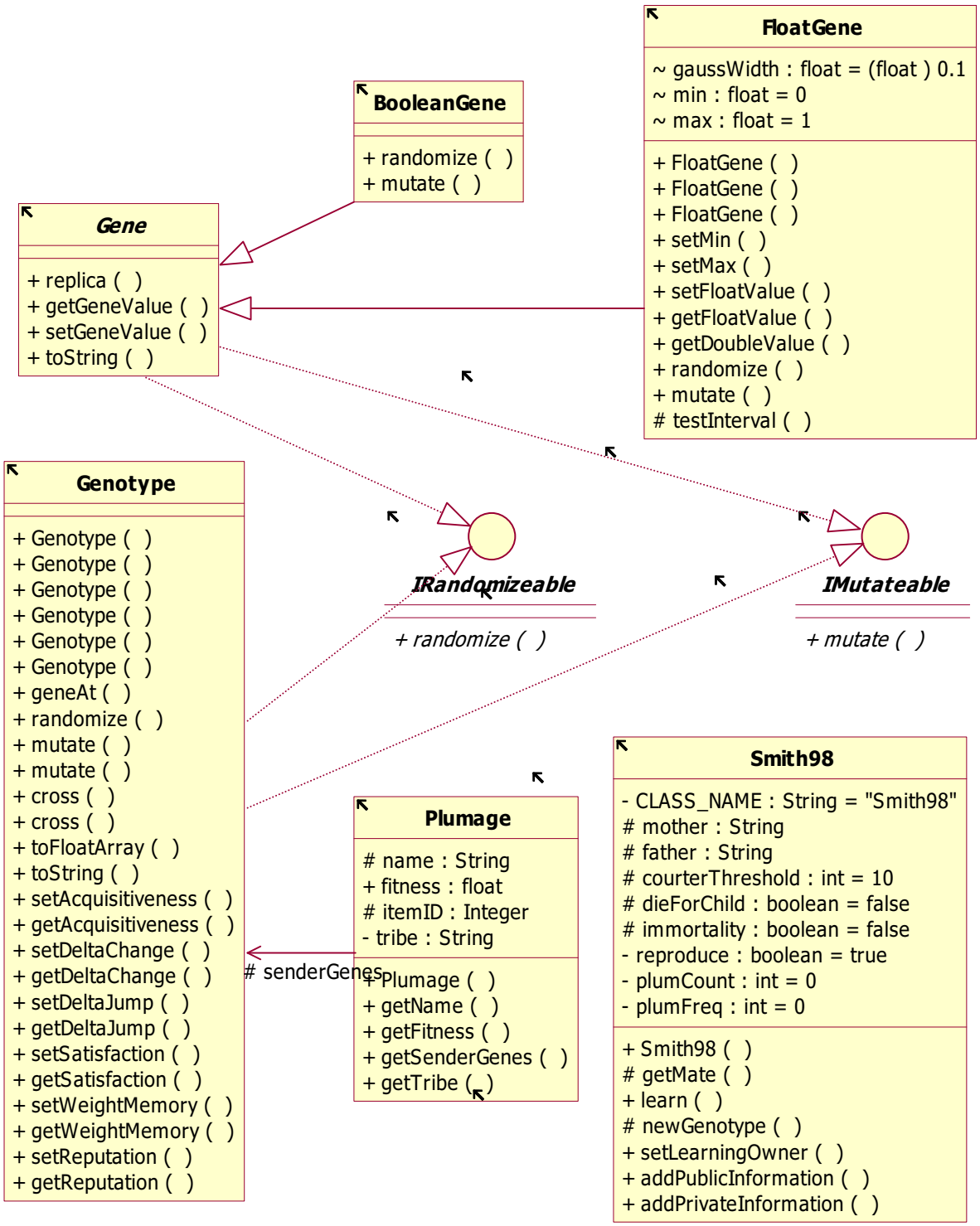


Figure 16. UML Class diagram of the learning algorithm

5 Mappable applications

This section compares the CATNETS market model, and the component modules identified so far, to existing applications with a great number of connected peers. All of these applications exhibit the unfavourable properties presented in the previous sections and require efficient coordination mechanisms. By introducing the CATNETS market model, we aim to ameliorating the performance.

Considering possible application domains, the following 3 systems have been identified: BITTORRENT, PLANETLAB and CORAL. They will be presented shortly and an analysis is given on the possible matching of Catallaxy to those applications. This matching will be done on the application layer. Lower Network layers are not explicitly mentioned.

5.1 BitTorrent

In the context of peer-to-peer networks, we have selected a P2P protocol which has a clearly specified protocol, that is popular enough, and that is used for clearly useful and legal purposes (some other P2P networks are almost only used for sharing copyrighted content). This protocol is BitTorrent (Cohen 2003). BitTorrent in general is introduced in deliverable D3.1; in this section we discuss the mapping to the CATNETS model.

A torrent consists of a central component, called tracker and all the current active peers (Izal, Urvoy-Keller et al. 2004). BitTorrent distinguishes between two kinds of peers depending on their download status: clients that have already a complete copy of the file and continue to serve other peers, called seeds; clients that are still downloading the file are called leechers. The tracker is the only centralized component of the system. The tracker is not involved in the actual distribution of the file; instead, it keeps meta-information about the peers that are currently active and acts as a rendez-vous point for all the clients of the torrent.
A user joins an existing torrent by downloading a torrent file (usually from a Web server), which contains the IP address of the tracker. Generic or specialized web search engines usually lead to pages where a file can be downloaded from one or several trackers. The user has to select one torrent file (and thus the tracker) to start downloading the file which will let him connect to the tracker and an initial seed with a complete copy of the file. In case of multiple trackers available for the same object, statistics about every tracker are published to help the visitor choose the right tracker. To update the tracker's global view of the system, active clients periodically (every 30 minutes) report their state to the tracker or when joining or leaving the torrent. Upon joining the torrent, a new client receives from the tracker a list of active peers to connect to.

In terms of the CATNETS model, people interested in downloading a file, running a web browser and a Bittorrent client has the role of a *Complex Service*. They look for a torrent file (a tracker) on several online web catalogues and look at the statistics of several trackers offering the same file. They manually select one tracker (the *Service market)*. The tracker adds the requesting complex service to a swarm of peers (LRMs) exchanging fragments of the file of common interest. All BitTorrent clients in the swarm belong to the *Resource market* and are acting as *Resources*.

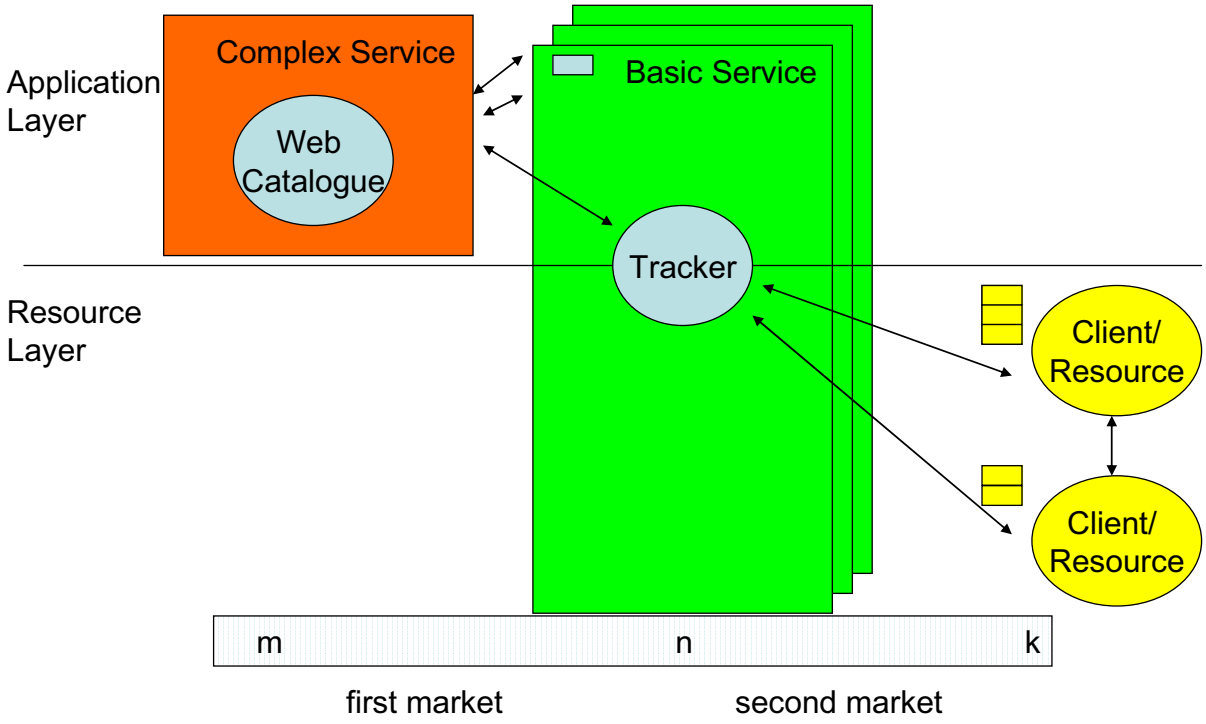


Figure 17. BitTorrent and CATNETS

5.2 PlanetLab

PlanetLab (Chun, Culler et al. 2003) is a geographically distributed overlay network designed to support the deployment and evaluation of planetary-scale network services. Two high-level goals shape its design. First, to enable a large research community to share the infrastructure, PlanetLab provides distributed virtualization, whereby each service runs in an isolated slice of PlanetLab's global resources. Second, to support competition among multiple network services, PlanetLab decouples the operating system running on each node from the network-wide services that define PlanetLab, a principle referred to as unbundled management.

The service-resource cycle in PlanetLab is as follows:

- In every node, the node manager is in charge of creating and allocating resources to vservers (virtual machines), and the resource monitor is in charge of tracking node's availability of resources and informing the central agent about available resources.

- The agent tracks nodes' free resources, which are advertised to resource brokers and offered as tickets to services interested in acquiring and using resources. This agent is part of PLC (Planet-Lab Central), a centrally-controlled brokerage service that can be decentralized using a delegation mechanism.

- In every service, the resource broker obtains tickets from agents on behalf of service managers, which are in charge of redeeming tickets with node managers to acquire resources, and if resources can be acquired, start the service in that node.

In terms of the CATNETS model, people or processes interested in using a given service have the role of *Client*. They should look for and select a service instance (a *Service Copy* in the *Service Market*). All nodes (represented by node managers and resource monitors; acting as *Resources*), all service instances (represented by resource brokers and service managers; acting as *Service Copies*), both mediated by the central Agent (PLC) belong to the *Resource market*.

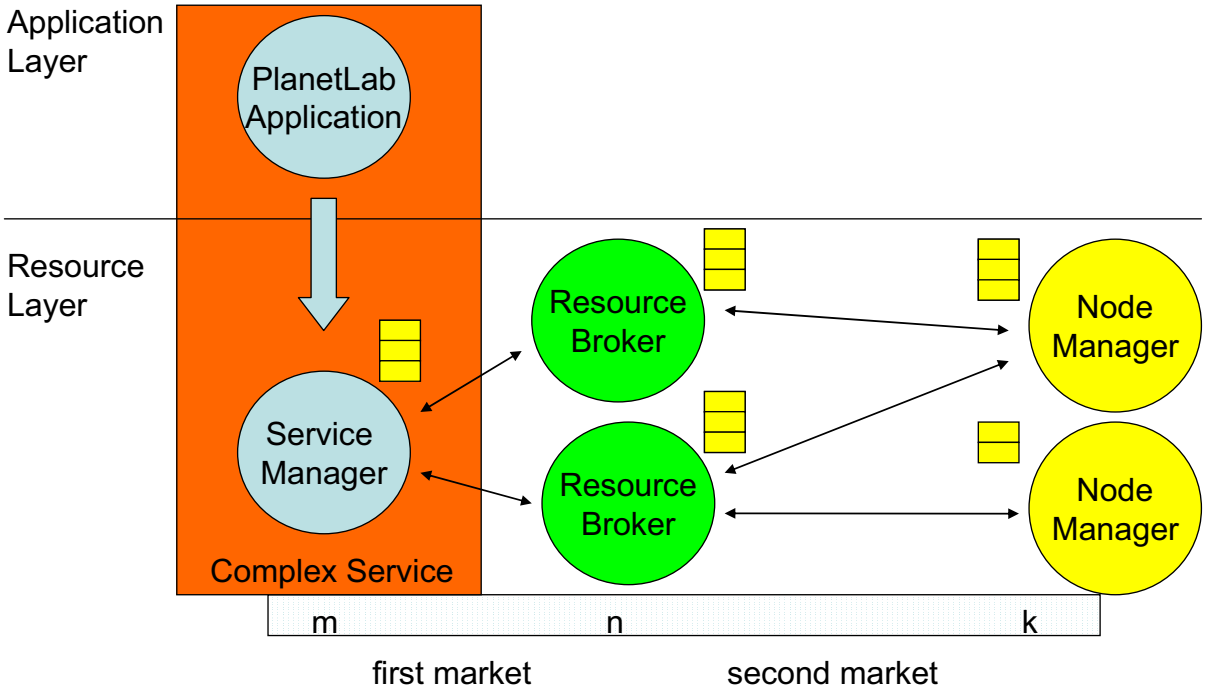


Figure 18. PlanetLab and CATNETS

5.3 Coral

Coral CDN (Freedman, Freudenthal et al. 2004) is a decentralized, self-organizing, peer-to-peer web-content distribution network (use illustration in figure 1.2). Coral CDN leverages the aggregate bandwidth of volunteers (typically PlanetLab slivers) running the software to absorb and dissipate most of the traffic of web sites using the system. In doing so, CoralCDN replicates content in proportion to the content's popularity, regardless of the publisher's resources, in effect democratizing content publication.

To use Coral CDN, a content publisher — or someone posting a link to a high-traffic portal — simply appends ".nyud.net:8090" to the hostname in a URL. Through DNS redirection, oblivious clients with unmodified web browsers are transparently redirected to nearby Coral web caches. These caches cooperate to transfer data from nearby peers whenever possible, minimizing both the load on the origin web server and the end-to-end latency experienced by browsers.

This requires two mechanisms: finding a close peer at first, then finding a close copy of the requested object. The first is achieved by mapping Coral servers and clients into clusters based on latency. The second is done using a locality-aware request routing algorithm or in-

dexing abstraction (also know as a Distributed Sloppy Hash Table or DSHT). Every Coral peer is running three elements: a DNS server, a HTTP proxy and a DSHT element.

In terms of the CATNETS model, people interested in downloading a file, running an unmodified web browser (or one with a Coral plug-in to "coralize" URLs) has the role of a *Client*. They request a "coralized" URL, thus going to a Coral DNS server where a response, the IP address of a close-by Coral proxy will be selected among many of them, based on the location of the client (this is the *Service market* and the Coral http proxy has the role of *Service Copy*). The client web browser will contact the http proxy with the given IP address. Then the proxy will look for the requested file in its own store or it will look for a close copy of the file in other peers using the Coral DSHT routing algorithm. Proxies belong to the *Resource market*, the election in the market is determined by the DSHT algorithm looking for a close copy of a file, and proxies are acting as *Resources*.

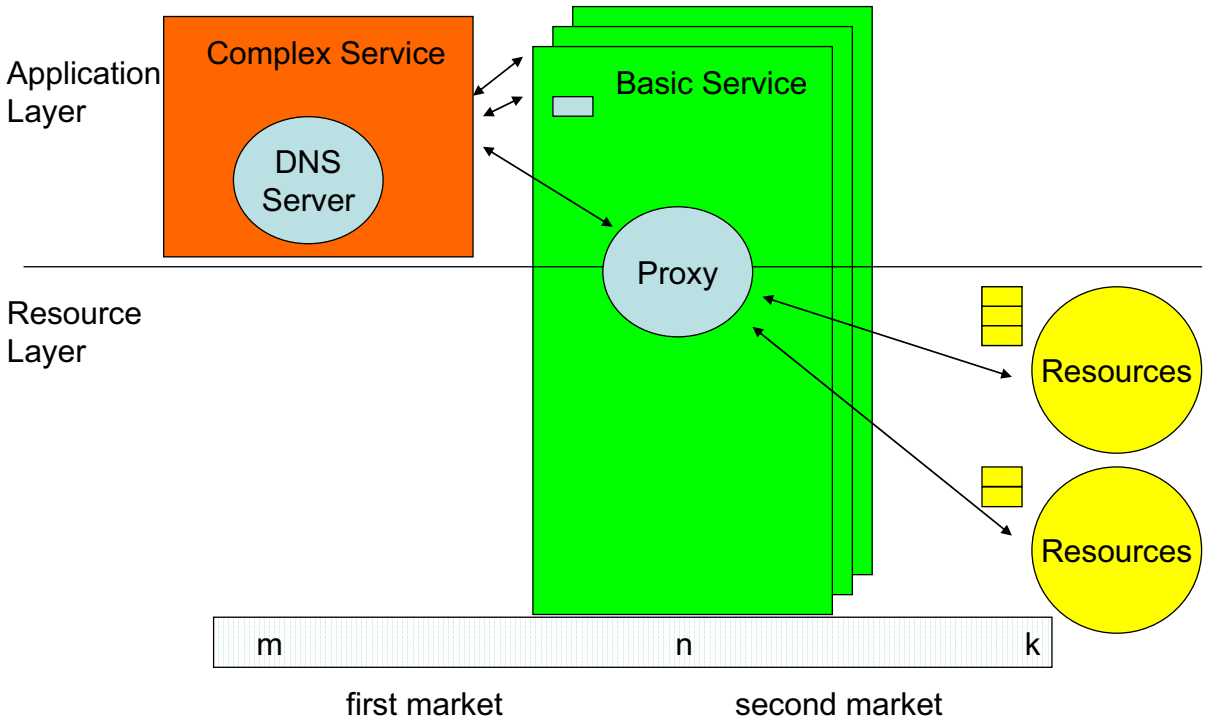


Figure 19. Coral and CATNETS

6 Summary

What makes Economics so attractive for computing environments is that its central research question lies in the effective allocation of resources, provided by suppliers and in demand by customers. In computing environments like Grid Computing, the resources in question are processor time or storage space, while the economic actors are computers or web services (Buyya, Abramson et al.; Buyya). It appears that, by just implementing markets in computing environments, the satisfying ability of economics might be viable for creating cost-effective computer architectures.

However, between the mostly descriptive economic concept and the normative technical implementation lies a fundamental gap, requiring selective choice of how actors, resources, goods, and markets are modelled and embedded in a technical environment. Some researchers call this task "market engineering" (Weinhardt, Holtmann et al.). The basic purpose of market engineering is to capture the inherently decentralized, dynamic coordination nature of the economic concept, and to translate that into a technical realization, which allows optimizing resource allocation.

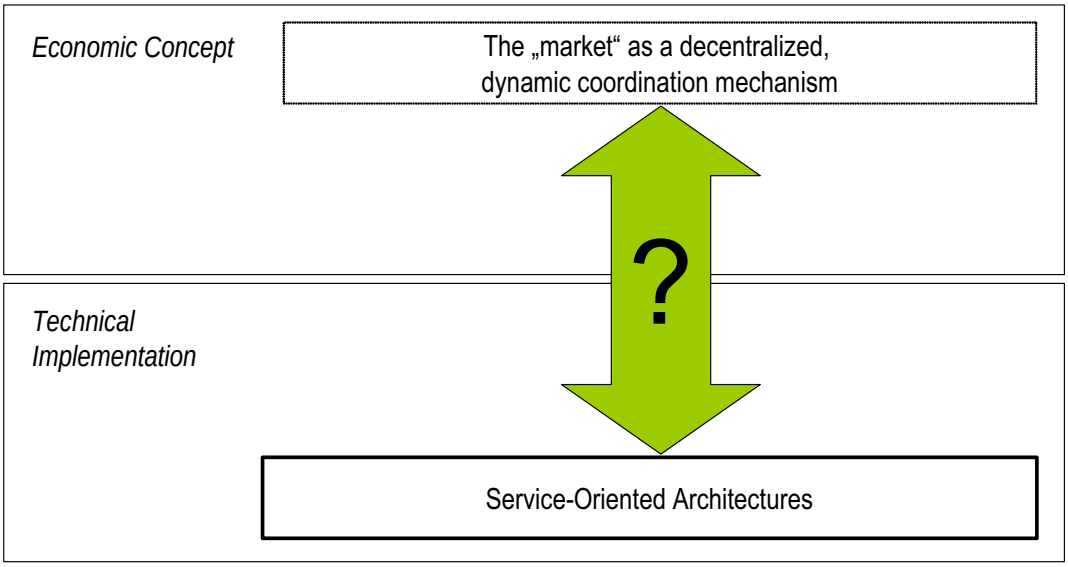


Figure 20: Realizing economic concepts in a technical implementation

There are several competing descriptive approaches to how economic resource allocation mechanisms work. In general, Economics is essentially all about the coordination of systems consisting of utility-maximizing agents, who satisfy their needs using some *mechanism* for solving a distributed resource allocation problem. The *effect* of this mechanism is a state where prices are set, so that supply and demand is perfectly balanced, and the number of transactions is maximized (Kearney, Smith et al.). All implementation attempts try either to recreate the mechanism, or to achieve the effect by using another mechanism, adding some side condition like zero communication costs or a steady environment state.

Adam Smith's proverbial *invisible hand* (Smith) was a first concept of a decentralised mechanism without a co-ordinator, but Smith gave no implementation of that mechanism. A century later, Leon Walras (Walras) introduced a central auctioneer, who iteratively solved the allocation problem out of total knowledge of supply and demand. With this mechanism, Walras was able to generate the desired equilibrium effect.

Most of today's economic research relies on Walras' *tatônnement* process as a valid picture of the mechanism, which influences also the possible realization in computing environments. An

example is the realization by Wellman (Wellman), titled Market-Oriented Programming (MOP), in an distributed artificial intelligence (DAI) environment. Wellman takes the notion that "an economy is a multiagent system" literally; the distributed agents individually compute their utility functions and post that information to a centralized Walrasian auctioneer. During the computation process, interrelated markets are successively brought to near-equilibrium by the auctioneer, with the final general equilibrium effect as the "gold standard" to achieve. MOP has been successfully used in electricity markets (Ygge), for multi-commodity flow problems (Wellman), supply chain management (Wellman and Walsh) or for negotiations about the quality of service in multimedia networks (Yamaki, Wellman et al.).

In contrast, Economics research on self-organization still aims at explaining the mechanism of the invisible hand, e.g. Agent-based Computational Economics (Tesfatsion). Actually, there is growing interest in using self-organization, as indicated by the start of large industrial research concepts like IBM's *Autonomic Computing* initiative. Autonomic Computing uses a biological paradigm as a design and control metaphor, the autonomic nervous system (Kephart and Chess). If the mechanisms underlying Hayek's *spontaneous order* concept (Hayek, Bartley et al.) can be properly understood, it might be possible to build large Autonomic information systems using the Catallaxy approach, where artificial entities coordinate themselves, just as human economy participants do in the real world. For a start, we have to discuss whether the desired effects of Autonomic Computing are achievable (and describable) using economic terms.

IBM's Autonomic Computing Manifesto (IBM) describes seven characteristics, which self-adapting systems should exhibit. The core characteristics are contained in the so-called CHOP cycle of self-configuring, self-healing, self-optimizing and self-protection capabilities. The *self-configuration* property is indicated in the variation of prices when adding or removing service providers (cf. the different density regimes). The *self-healing* of the system is apparent in case a service provider instance shuts down or a network connection gets broken (cf. the different dynamics regimes). The application is *self-optimizing*, in that the agents constantly attempt to change their strategies towards the maximum utility-eliciting negotiation positions, which respectively lay on the total supply and demand curves. The *self-protection* of the application finally can be reached by including security mechanisms like reputation tracking (Eymann, Padovan et al.), which are effective in separating malicious and underperforming agents.

In addition, viewing Autonomic Computing systems as Economic systems has some merits, too. The main applications for AC systems will be deeply rooted in a business context. With a biological background, you need to find biological translations for conceptual data structures and functionality for describing success, utility, or business goals. This is not a trivial process, and may lead to semantic loss underway. For example, the business goal of maintaining availability (to prevent loss of profit in the case of server downtime), may be translated biologically as "staying alive". However, the semantics of both differ – deliberately shutting down a biological AC system may qualify for murder, while in economic terms, shutting down a system means buying it out of business – with the programmer defining what the currency is.

The key to this semantic shift is to view the "market" as an emergent mechanism of coordinating and matching supply and demand offers, and market participants as rationally-bounded, self-interested individuals, like in Neo-Austrian (Hayek, Bartley et al.) or Neo-Institutional Economics (North, Calvert et al.)(Furubotn and Richter). As we move on to technology, which allows us to map unstructured semantic knowledge in large and very dynamic systems, we have to look for decentralized self-organization, not at least for reasons of exponentially increasing "costs of ownership" (Truex, Baskerville et al.).
Given a highly complex and dynamic ALN infrastructure, scalability and the management of a great number of heterogeneous resources are supposed to be challenging issues of future

ALNs and computing systems in general. Ubiquitous computing (Weiser) envisions trillions of computing devices connecting and interacting with each other; Grid Computing (Snelling and Priol) envisions millions of networked processors. To handle the complexity and scale of such systems, the necessity of a centralized management could easily turn the vision into a "nightmare" (Kephart and Chess). The solution is not necessarily a question of overcoming semantic gaps or problems of multi-attribute optimization. Discovering and selecting web services from huge numbers of unreliable candidates alone is challenging enough.

7 Appendix: Documentation of CatNet.v1

7.1 Components Code

A preliminary evaluation of "Catallactic" mechanisms in the FET assessment project CatNet (IST-2001-34030) using computer simulations has shown positive results, upon which the CATNETS project will build. The CatNet assessment project (IST-2001-34030) has conducted several simulation experiments using Catallactic service selection in an ALN model. The performance of the mechanism has been evaluated using both economic and computer science benchmark parameters (CatNet Project 2002; CATNET Project 2003a; CATNET Project 2003b).

We have implemented two main control mechanisms for the network coordination: the baseline and the catallactic control mechanism. The baseline mechanism computes the service/resource allocation decision in a centralized instance, using a closed first-price auction. In the catallactic mechanism, autonomous agents take their decisions in a decentralized way, having only local information about the environment. Each agent uses a strategy to take decisions, which targets to increase the agent's own benefit.

The schema below shows the catallactic market, developed formerly in the assessment project, which does not comprise the same detailing like the derived models depicted above:
The client contains a buyer interface on the service market and the service has a seller same implementation for the service market and the resource market. The resource has a seller interface on the resource market. In this market model it is not possible for clients to contract resources directly. The two stage market model is kept. If a client would like to buy a resource then there must be a service contracted beforehand that will provide this resource to the client.

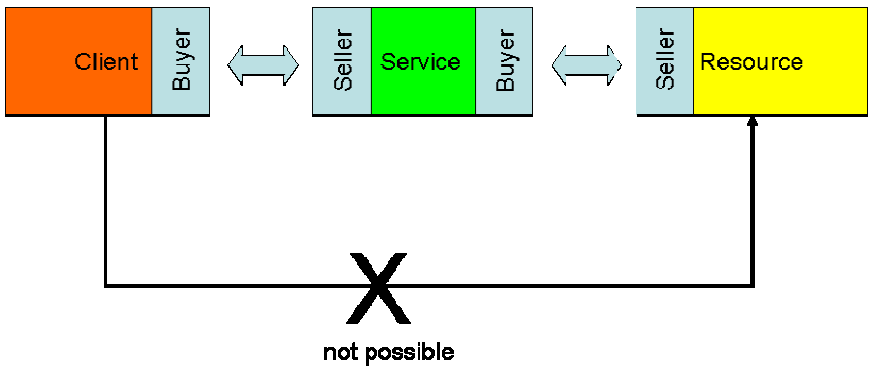


Figure 21. Decentralized market model in CatNet

In the centralized market a central matchmaker matches the service requests of the first market and the resources of the second market. This market model must also take into account that a client cannot buy a resource directly.

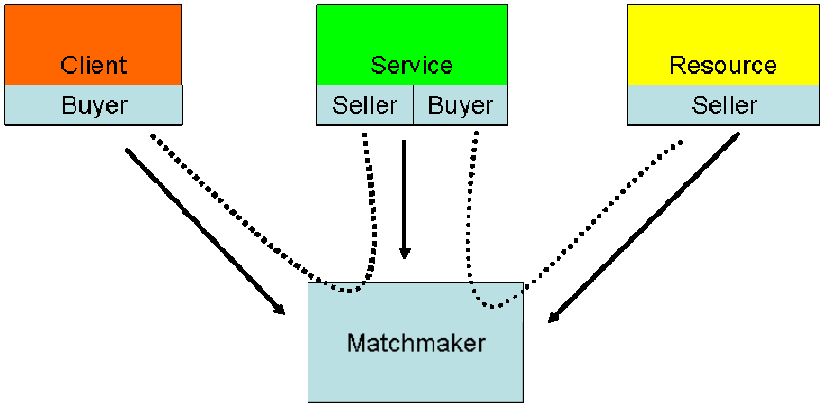


Figure 22. Centralized market model in CatNet

In general, we have found that the catallactic model becomes relatively superior over the baseline model with increasing dynamics and increasing density, as indicated by the social welfare utility (SWF) criteria ardaiz (Ardaiz, Freitag et al. 2002). But Catallaxy achieves this result at the expense of higher communication cost (CC) and inferior response times (REST). This is caused by the overhead of the negotiation protocol, the multiple bilateral communications, and the varying distance between service copies and clients. These results, which are consistent in all of the investigated parameters, are described in detail in the corresponding deliverable of the CATNET project (CATNET Project 2003a).

The initial exploration conducted in the CATNET assessment project has shown in a simulated environment that service provision in application layer networks can be coordinated by decentralized "catallactic" mechanisms. We observed the robustness of catallactic coordination against increasing dynamics of networks, which potentially is a highly interesting feature for heterogeneous and ad-hoc Grid and P2P networks. The simulation results also raised new questions, which could not be answered in the scope of the assessment project, due to the abstraction level of the simulator and the absence of real application data from existing networks. The purpose of CATNETS is to explore this effect deeper, both by simulation and by including real application data.

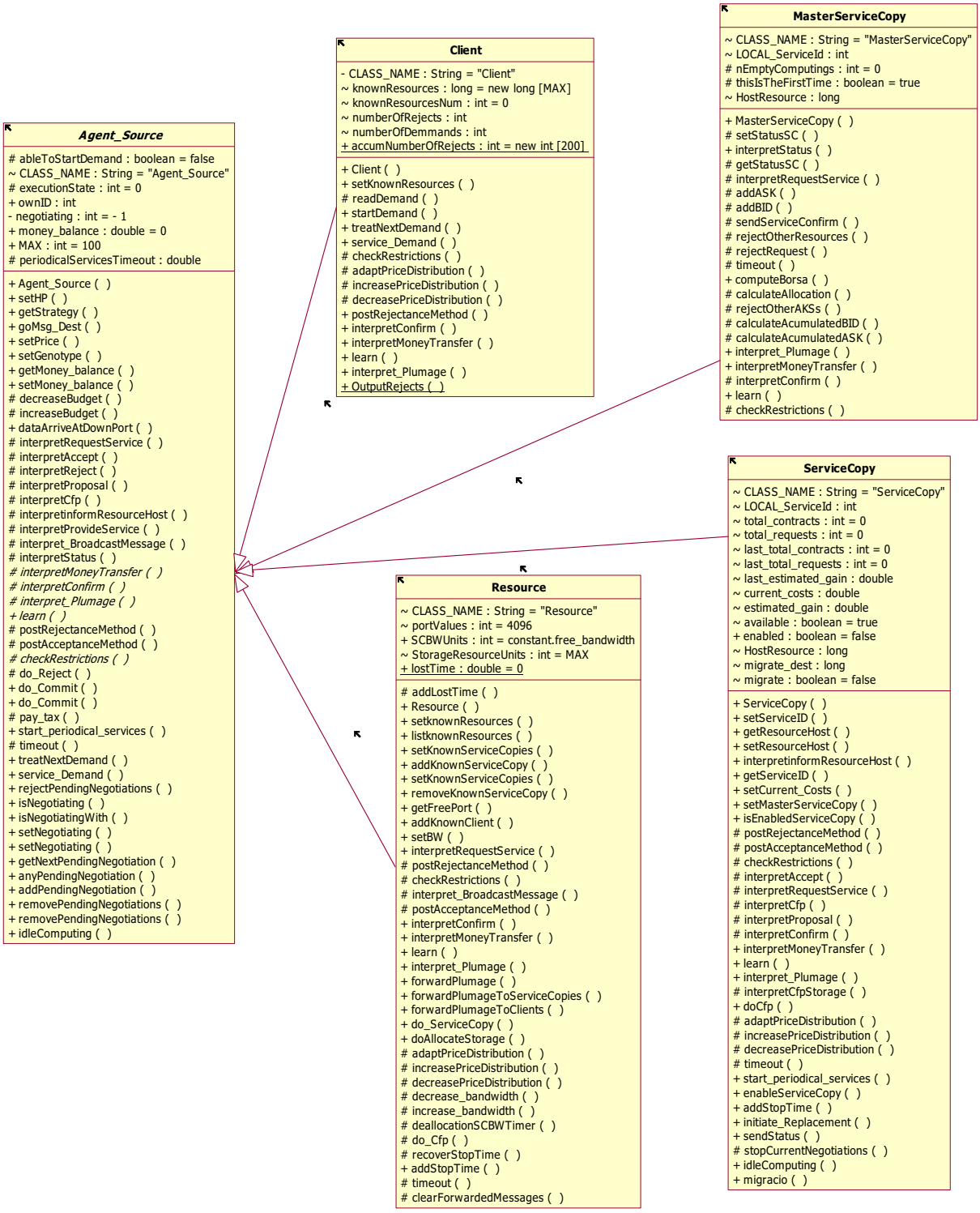


Figure 23. Classdiagramm CatNets.v1

7.2 Simulation Environment and Documentation

The CATNET simulator has been implemented on top of the JavaSim (Breslau et al. 2002; JavaSim Project 2002) network simulator. It can be configured to simulate a specific ALN, such as a content distribution network or peer-to-peer network. Different agent types can be instantiated, namely clients, resource agents, and service agents. Network resources to be allocated encompass service access, bandwidth and storage. The simulation builds on a TCP/IP

network model supported by JavaSim. It describes the generic structure of a node and the generic network components, which can both be used as base classes to implement protocols across various layers.

The CatNet Simulator can be installed on any computer system running on a Unix Operating System (e.g. Suse/Linux 9.0; Kernel 2.4.21-215-smp4G / i686). However, it is recommended to install the simulator on a computer cluster or a multi-processor machine, as the experiments – running all at the same time – require lots of processor power. Though, it is possible starting only one experiment (out of 50) on an ordinary desktop machine, but the results can only be evaluated when all 50 experiments are completely finished, as the evaluation scripts require availability of all (raw) results. Thus, a sequential processing of the simulations can be performed on a desktop machine, but it is time consuming.

To install the CatNet software, some additional software has to be installed.

- Tcl/Tk must be installed to run the simulation scripts (version 8; http://www.tcl.tk/).
- Java SDK, version 1.4.2 from http://java.sun.com/j2se/1.4.2/download.html
- J-Sim is downloadable from http://www.j-sim.org/. We were working with version 1.3. The installation of the patches is not necessarily needed. The package comes with the binaries, thus it is not necessary to compile the sources, which sometimes leads to errors due to incompatibilities of the java version and the J-Sim version.

After that, you need to download and install the java code of CatNet.v1. This could be done by downloading from the CVS.

cvs access mode: pserver
Servername wi.oec.uni-bayreuth.de
repository /home/cvsroot

-> CVSRoot :pserver:wi.oec.uni-bayreuth.de:/home/cvsroot

Install all files in one root directory "catnet". Your catnet directory now should look like this:

-classes
-jars
-jsim-1.3
-script
-setPath.sh
-src
-tcl

Figure 24. Directory listing

The directory *classes* contains the java classes for *catnet*, *jars* some necessary libraries (like database support), *script* the configuration scripts respectively run scripts and log-files, *src* contains the sources of *catnet* and finally *tcl* the Tcl/Tk Environment.

setPath.sh is a shell script to set variables to the environment and must be run before starting the simulations.

```
export JAVA_HOME=/usr/java;
export JAVASIM=/home/reinickm/catnet/jsim-1.3;
export CATNET=/home/reinickm/catnet;
```

```
export PATH=$JAVA_HOME/bin:.:$PATH;
export script=$JAVASIM/jars/tcl.zip:$JAVASIM/jars/jython.jar;
export CLASSPATH=$CLASSPATH:.:$CATNET/classes:$JAVASIM/classes:$JAVASIM/jars
/tcl.zip:$JAVASIM/jars/jython.jar;
export JAVA_DEV_ROOT=$CATNET;
```

Adjust line 1 to 3 to your local directories. And then run the script:
"$ source setPath.sh"

Now, the variables are set in the unix shell environment (for the current session).

Topology and Agents
Figure 26 shows the simulation topology, consisting of 106 nodes per ring. Each node is a resource, being at least responsible for message forwarding. On each node, a client or service copy could be instantiated. Clients are exclusively located on the outer leaves, whereas service copies are distributed randomly to the existing nodes/resources by the start-up scripts. For the experimentation, 75 clients and 50/30/25/12/6 (depending on the density level) service copies are instantiated; the location of the service copies is depending on the start-up scripts (see next section).
The one ring topology can be extended to a two or three ring topology (see Figure 26), however the position of the Master Service Copy will neither change nor does it duplicate its capabilities.

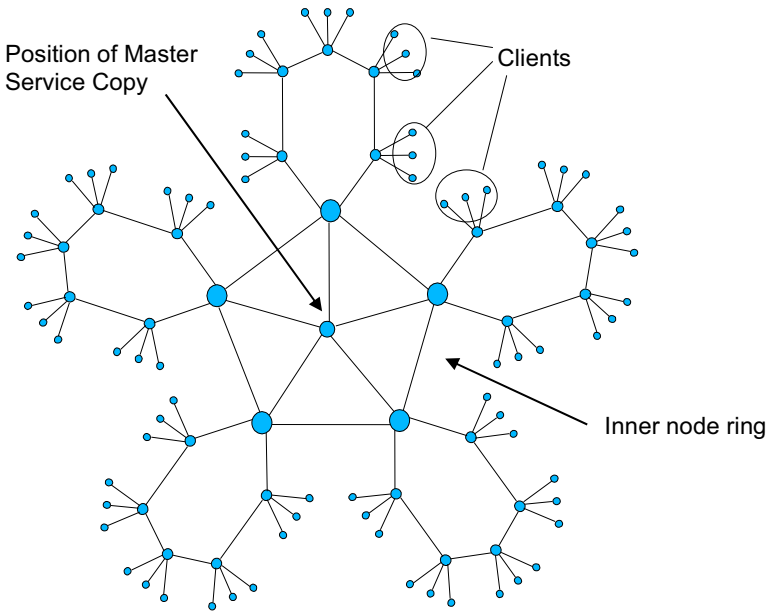


Figure 25. Simulation Topology (1 ring)

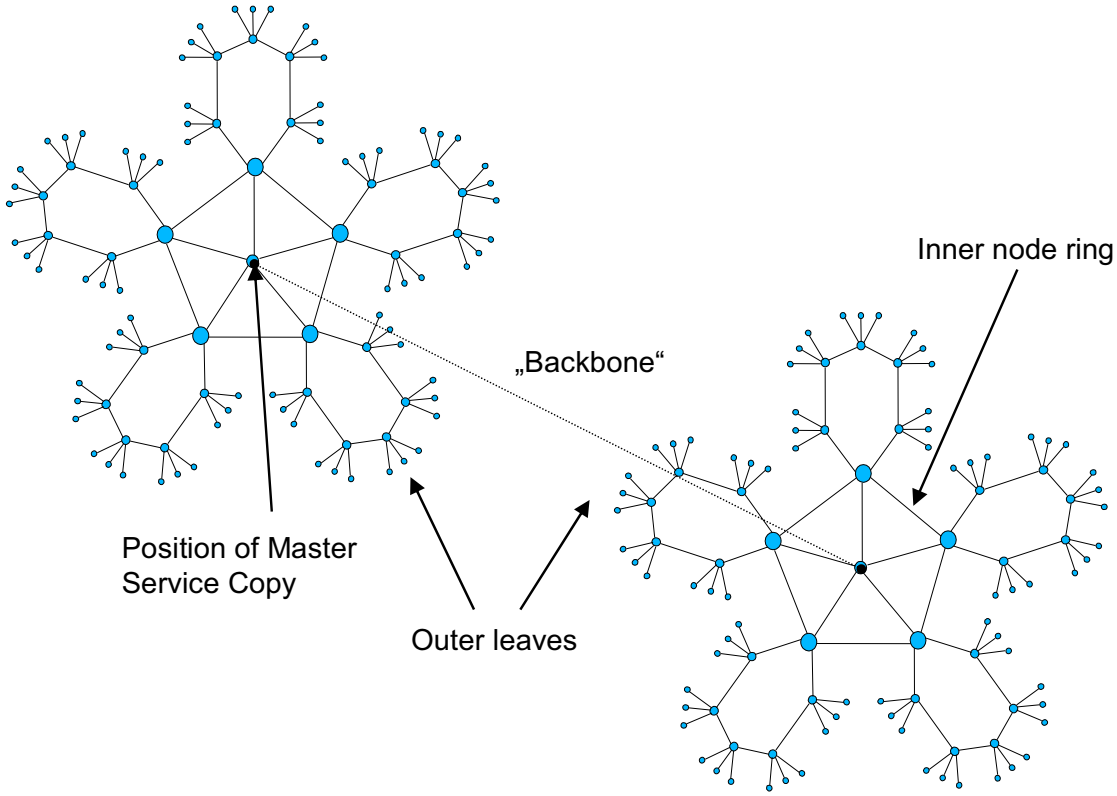


Figure 26. Simulation Topology (2 rings)

Scripts

The experiments are performed to measure the behaviour of the approaches on behalf of service density and dynamics. Testing 5 density levels and 5 dynamicity levels leads to 25 runs, for catallactic and baseline approach these sums up to 50 simulation runs. The concerned density and dynamicity level for the current simulation run is set by the start-up scripts. In the following section an exemplified script call for one simulation run is described.

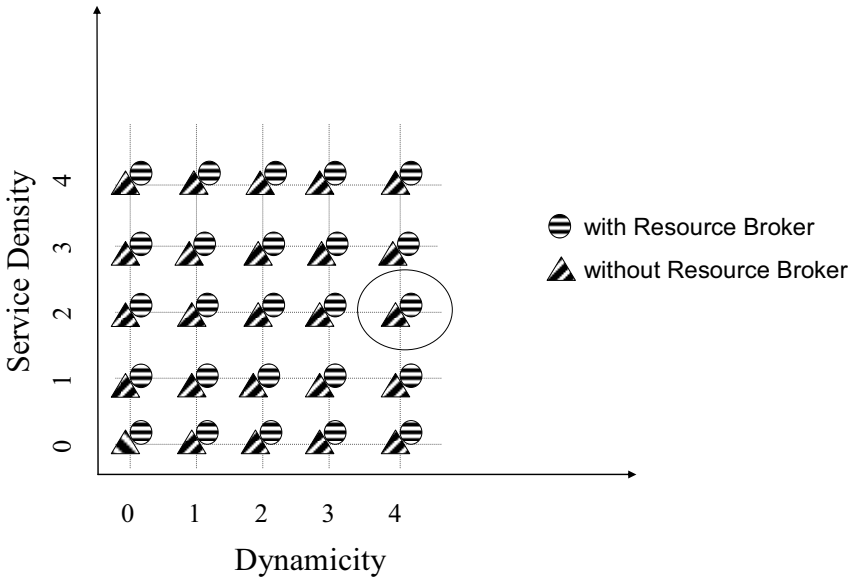


Figure 27. Simulation Scenarios

For simplicity reasons, exclusively the simulation run with density 2, dynamicity level 4 in a single ring topology shall be executed (Figure 25). This can be accomplished by calling *$ exp25b42_1.sh &* in the directory *scripts* which runs the simulation by calling *java drcl.ruv.System -ue exp_25b_dyn4_den2_an1.tcl >./logs/exp25b42.log.* This command is the most important in the concerned script; the following 4 lines are only for pre-calculating some metric values.

```
$ more exp25b42_1.sh
#!/bin/bash
java drcl.ruv.System -ue exp_25b_dyn4_den2_an1.tcl >./logs/exp25b42.log
cat logs/exp25b42.log | grep "Graph output - confirm" >./logs/exp25b42_graph.log
java catnet.util.TimeComputeRAEFromFile ./logs/exp25b42_graph.log ./logs/exp25b42_rae.log 1000 2000
grep CC: ./logs/exp25b42.log | cut -f 2,3 >./logs/exp25b42_netinfo.log
java catnet.util.TimeComputeNetInfoFromFile ./logs/exp25b42_net.log ./logs/exp25b42_netinfo.log 1000 2000
```

exp_25b_dyn4_den2_an1.tcl launches the start-up scripts depending on the scenario (level of density and dynamicity). The following scripts are called successively from exp_25b_dyn4_den2_an1.tcl: First call is exp_25_common.tcl.

```
$ more exp_25b_dyn4_den2_an1.tcl
source topologies/exp_25_common.tcl
source topologies/exp_dyn4.tcl
java::call catnet.data.constant setBaselineOn
source topologies/exp_25_topo_an1.tcl
source recursos/exp_25_rec.tcl
source recursos/exp_25_rec_den2.tcl
set netlogfilename "exp25b42_net.log"
set demandfilename "exp25_demmand.txt"
source demandes/exp_25_dem.tcl
```

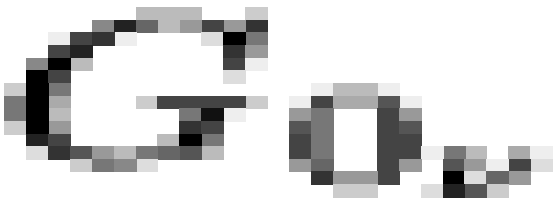
exp_25_common.tcl sets a punch of variables in the java class catnet.data.constant. Shown are the regularly used values like allocation cycle time of the Master Service Copy (only used for baseline), setLearning switches the learning algorithm on or off etc.

```
$ more exp_25_common.tcl
java::call catnet.data.constant setAllocationTime 0.15
java::call catnet.data.constant setLearningOff
#40 20 10 5
java::call catnet.data.constant setStockMarcketOn
java::call catnet.data.constant setHopFactor 30
java::call catnet.data.constant setUpdatesOn
java::call catnet.data.constant setMSCUpdateTime 2000
#java::call catnet.data.constant setMigrationOn
```

Right after exp_25_common.tcl, exp_dyn4.tcl is called to set the dynamicity level. Dynamics are implemented exclusively on the service copies. Each instance that could fail (service copies) periodically generates a random value and checks it against a given probability. The example initializes such a period of 200ms, whereas 50% of the service copies are initially switched off and the percentage to check against whether to change the state of the instance is 40%. These values can be changed to adapt the dynamicity levels.

```
$ more exp_dyn4.tcl
java::call catnet.data.constant setDynamicParameters 50 40 200
```

exp_25_topo_an1.tcl at first sets some variables for constructing the network and sets the topology of 1 ring by means of an adjacency matrix (section $adjMatrix_ set). The adjacency matrix describes the links between nodes, 1 indicates that a link exists. It is set with the command
*java::call drcl.inet.InetUtil createTopology [! .] "n" "n" $adjMatrix_ $link_*



Then, it attaches protocol stacks and attaches the agents to the nodes (section

```
udp                         drcl.inet.transport.UDP
resource          101/udp  catnet.resource.Resource
client             102/udp  catnet.client.Client
servicecopy103 103/udp  catnet.servicecopy.ServiceCopy).
```

The numbers 101 et seqq. set the udp-ports which are monitored for incoming messages. After that, setknownResources sets the addresses of the neighbouring nodes to allow message routing.
Finally it sets the links, generated from the adjacency matrix (java::call drcl.inet.InetUtil setupRoutes [! .] $adjMatrix_).

```
$ more exp_25_topo_an1.tcl
puts "Constructing network"

cd [mkdir drcl.comp.Component /test]
```

```
#puts "Define and open database..."
#Syntax: opendatabase "host" "username" "password" "Name of Experiment"
#java::call catnet.log.store opendatabase "drewpc2.iig.uni-freiburg.de/CatNet_Data" "cn" "CatNet" "Cat_01"

puts "Create topology..."

set link_ [java::new drcl.inet.Link]
$link_ setPropDelay 0.01

# Variables
set level1 5
set level2 6
set nclients 3
set nscs 1
set anillos 1
set level3 [expr $nclients]
set numnodes [expr $level1 + (($level2)-1)*$level1 + $level1*(($level2)-1)*$level3]

# adjacency matrix
set adjMatrix_ [java::new int\[\]\[\] [expr ($anillos * ($numnodes + 1))]]
#set adjMatrix [java::call drcl.inet.InetUtil.getAdjMatrixFromFile "test.brite"]

set currlvl2 [expr $level1]
set currlvl3 [expr $level1*$level2]
for  {set l 0} {$l < $anillos} {incr l} {
$adjMatrix_ set [expr ($l * ($numnodes + 1)) + $numnodes] [java::new int\[\] [expr ($level1+2)]]
 for {set i 0} {$i<$level1} {incr i} {

  #if {$l > 0} {
    $adjMatrix_ set [expr $i + ($l * ($numnodes + 1))] [java::new int\[\] 5]
  #}
  #$adjMatrix_ set [expr $i] [java::new int\[\] 5]
  [$adjMatrix_ get [expr $i + ($l * ($numnodes + 1))]] set 0 [expr (($i + 1)%$level1) + ($l * ($numnodes + 1))]
  [$adjMatrix_ get [expr $i + ($l * ($numnodes + 1))]] set 1 [expr (($i - 1)%$level1) + ($l * ($numnodes + 1))]
  [$adjMatrix_ get [expr $i + ($l * ($numnodes + 1))]] set 2 [expr ($level1 + $i*($level2-1)) + ($l * ($numnodes
+ 1))]
  [$adjMatrix_ get [expr $i + ($l * ($numnodes + 1))]] set 3 [expr ($level1 + ($i + 1)*($level2-1) -1) + ($l *
($numnodes + 1))]

  # Set adjacency matrix for the cental pentagons
  [$adjMatrix_ get [expr $i + ($l * ($numnodes + 1))]] set 4 [expr $numnodes + ($l * ($numnodes + 1))]
  [$adjMatrix_ get [expr $numnodes + ($l * ($numnodes + 1))]] set [expr $i] [expr $i + ($l * ($numnodes + 1))]

   # Set adjacency matrix for the subpentagons   if {$l > 0} {
    [$adjMatrix_ get [expr $numnodes + ($l * ($numnodes + 1))]] set 5 [expr $numnodes + ($l-1) * ($numnodes
+ 1)]
    [$adjMatrix_ get [expr $numnodes + ($l-1) * ($numnodes + 1)]] set 6 [expr ($numnodes + ($l * ($numnodes
+ 1)))]
   }

  #Set adjacency matrix between the central pentagons
  for {set j 0} {$j<($level2 - 1)} {incr j} {
    $adjMatrix_ set [expr $currlvl2 + ($l * ($numnodes + 1))] [java::new int\[\] [expr 2 + $level3]]
    # Si es el primer nodo del subpentagono
    if {$j == 0} {
      [$adjMatrix_ get [expr $currlvl2 + ($l * ($numnodes + 1))]] set 0 [expr $i + ($l * ($numnodes + 1))]
    } else {
      [$adjMatrix_ get [expr $currlvl2 + ($l * ($numnodes + 1))]] set 0 [expr ($currlvl2 - 1) + ($l * ($numnodes +
1))]
    }
    # If it is the last node of the subpentagon
```

```
    if {$j == ($level2 - 2)} {
      [$adjMatrix_ get [expr $currlvl2 + ($l * ($numnodes + 1))]] set 1 [expr $i + ($l * ($numnodes + 1))]
    } else {
      [$adjMatrix_ get [expr $currlvl2 + ($l * ($numnodes + 1))]] set 1 [expr $currlvl2 + 1 + ($l * ($numnodes +
1))]
    }

    #Set adjacency matrix of the vertices of the subpentagons
    for {set k 0} {$k<$level3} {incr k} {
      [$adjMatrix_ get [expr $currlvl2 + ($l * ($numnodes + 1))]] set [expr 2 + $k] [expr $currlvl3 + ($l * ($num-
nodes + 1))]
      $adjMatrix_ set [expr $currlvl3 + ($l * ($numnodes + 1))] [java::new int\[\] 1]
      [$adjMatrix_ get [expr $currlvl3 + ($l * ($numnodes + 1))]] set 0 [expr $currlvl2 + ($l * ($numnodes + 1))]

      set currlvl3 [expr $currlvl3 + 1]
    }

    set currlvl2 [expr $currlvl2 + 1]
  }
 }
  #Set parameters
  set currlvl2 [expr $level1]
  set currlvl3 [expr $level1*$level2]
  puts "Ring [expr $l+1] set"
}

java::call drcl.inet.InetUtil createTopology [! .] "n" "n" $adjMatrix_ $link_

puts "Building..."

set nb_ [mkdir drcl.inet.NodeBuilder .nb]
$nb_ setBandwidth 1000000; # 10Mbps

$nb_ build [! n*]

set starth [expr $level1*$level2]

for {set l 0} {$l < $anillos} {incr l} {
set downerlimit [expr $l * ($numnodes + 1)]
 for {set i $downerlimit} {$i<($starth + $downerlimit)} {incr i} {
 #puts "Set resources and servicecopies on node $i"
    $nb_ build [! n$i] {
        udp                     drcl.inet.transport.UDP
        resource  101/udp       catnet.resource.Resource
        servicecopy103 103/udp catnet.servicecopy.ServiceCopy
    }
 }
}

#puts "done"

# Put clients on the outer leaves
for {set l 0} {$l < $anillos} {incr l} {
set downerlimit [expr $starth + ($l * ($numnodes + 1))]
 for {set i $downerlimit} {$i<($numnodes + $downerlimit - $starth)} {incr i} {
  #puts "Set clients&ressources&SCs on node $i"
  $nb_ build [! n$i] {
        udp                     drcl.inet.transport.UDP
        resource  101/udp       catnet.resource.Resource
        client 102/udp          catnet.client.Client
        servicecopy103 103/udp catnet.servicecopy.ServiceCopy
  }
```

```
 }
}


 puts "SET MSC on node $numnodes"
 $nb_ build [! n$numnodes] {
  udp                drcl.inet.transport.UDP
  MSC1 101/udp       catnet.servicecopy.MasterServiceCopy
  #MSC2 102/udp              catnet.servicecopy.MasterServiceCopy
 }

puts "Set HPs of Ressources und SCs"
for {set l 0} {$l < $anillos} {incr l} {
 # Para los resources internos
 set downerlimit [expr ($l * ($numnodes + 1))]
 for {set i $downerlimit} {$i<($starth + $downerlimit)} {incr i} {
 #puts "Set HP on node $i, ress&SC"
   ! n$i/resource setHP $i 101
   ! n$i/servicecopy103 setHP $i 103
 }
}

puts "Set HPs of Clients und SC103s"
for {set l 0} {$l < $anillos} {incr l} {
 set downerlimit [expr ($l * ($numnodes + 1))]
 for {set i [expr $starth + $downerlimit]} {$i<($numnodes + $downerlimit)} {incr i} {
 #puts "Set HP on node $i, ress&client&SCs"
   # Client
   ! n$i/resource setHP $i 101
   ! n$i/client setHP $i 102
   ! n$i/servicecopy103 setHP $i 103
 }
}

puts "Set MSC HPs on node [expr $numnodes]"
#! n$i/MSC1 setHP [expr $numnodes] 101
#! n$i/MSC2 setHP [expr $numnodes] 102

! n$numnodes/MSC1 setHP [expr $numnodes] 101
#! n$numnodes/MSC2 setHP [expr $numnodes] 102

puts "Setup broadcast routes ..."

for {set l 0} {$l < $anillos} {incr l} {

 set currlvl2 [expr $level1 + $l * ($numnodes + 1)]
 set currlvl3 [expr $level1*$level2 + $l * ($numnodes + 1)]


 for {set i [expr 0 + $l * ($numnodes + 1)]} {$i< $level1 + $l * ($numnodes + 1)} {incr i} {
  #puts "i: $i | numnodes: $numnodes | currlvl3: $currlvl3 | currlvl2: $currlvl2 | level1: $level1"
  if {$l == 0} {
     ! n$i/resource setknownResources [expr ($i - 1)%$level1]
     ! n$i/resource setknownResources [expr ($i + 1)%$level1]
     puts "Hauptpentagon $i: kennt $currlvl2 | [expr $currlvl2 + $level2 - 2] | [expr ($i - 1)%$level1] | [expr ($i +
1)%$level1]"
      } else {
     ! n$i/resource setknownResources [expr ($i - 1 - $l * ($numnodes + 1))%$level1 + $l * ($numnodes + 1)]
     ! n$i/resource setknownResources [expr ($i + 1 - $l * ($numnodes + 1))%$level1 + $l * ($numnodes + 1)]
     puts "Hauptpentagon(l > 0) $i: kennt $currlvl2 | [expr $currlvl2 + $level2 - 2] | [expr ($i - 1 - $l * ($numnodes
+ 1))%$level1 + ($l * ($numnodes + 1))] | [expr ($i + 1 - $
l * ($numnodes + 1))%$level1 + $l * ($numnodes + 1)]"
```

```
    };
  ! n$i/resource setknownResources [expr $currlvl2]
  ! n$i/resource setknownResources [expr $currlvl2 + $level2 - 2]
#  ! n$i/resource setknownResources [expr $numnodes]
#  ! n$numnodes/resource setknownResources [expr $i]

  for {set j 0} {$j<($level2 - 1)} {incr j} {
    set temp [expr ($currlvl2)+1]
    # Last node of subpentagon
    if {$j == ($level2 - 2)} {
    #puts "Last node $currlvl2 kennt: $i | [expr $currlvl2-1]"
      ! n$currlvl2/resource setknownResources [expr $i]
      ! n$currlvl2/resource setknownResources [expr $currlvl2 - 1]
    # First node of subpentagon
    } elseif {$j == 0} {
    #puts "first node $currlvl2 kennt: $i | [expr $currlvl2 + 1]"
      ! n$currlvl2/resource setknownResources [expr $i]
      ! n$currlvl2/resource setknownResources [expr $currlvl2 + 1]
    # Other nodes of the subpentagon
    } else {
    #puts "Node $currlvl2 knows: [expr $currlvl2+1] | [expr $currlvl2-1]"
      ! n$currlvl2/resource setknownResources [expr ($currlvl2) - 1]
      ! n$currlvl2/resource setknownResources [expr ($currlvl2) + 1]
    }

    # Build routes between nodes on the outer leaves
    for {set k 0} {$k<$level3} {incr k} {
      ! n$currlvl3/resource setknownResources [expr $currlvl2]
      ! n$currlvl2/resource setknownResources [expr $currlvl3]
     #puts "last node: $currlvl3 knows $currlvl2 and $currlvl2 knows $currlvl3"
      set currlvl3 [expr $currlvl3 + 1]
    }
    set currlvl2 [expr $currlvl2 + 1]
  }
 }
}

# Build routes between central nodes
for {set l 0} {$l<$anillos} {incr l} {
set downerlimit [expr $l * ($numnodes +1)]
 for {set i $downerlimit} {$i < $level1 + $downerlimit} {incr i} {
 if {($l - 1) >= 0} {
 #puts "$i kennt [expr $i - $numnodes - 1]"
  ! n$i/resource setknownResources [expr $i - $numnodes - 1]
  };
 if {($l + 1) < $anillos} {
 #puts "$i kennt [expr $i + $numnodes +1]"
  ! n$i/resource setknownResources [expr $i + $numnodes +1]
  }
 }
}


puts "Setup static routes ... (May take a while)"
#for {set i 0} {$i<($anillos*$numnodes+1)} {incr i} {
#for {set j [expr $i+1]} {$j<($anillos*$numnodes+1)} {incr j} {
# puts "$i, $j"
#  java::call drcl.inet.InetUtil setupRoutes [! n$i] [! n$j] "bidirection"
  #  puts "[expr $i] -> [expr $j]"
#}
#if {$i % 10 == 0} {
#puts "up to Node [expr $i] set"
```

```
# }
#}

java::call drcl.inet.InetUtil setupRoutes [! .] $adjMatrix_
puts "Static routes set up"
```

exp_25_rec.tcl sets some attributes and values to the created resources and service copies, e.g. initial prices for services and resources, serviceIDs to identify the task of a service, budget and the address of the Master Service Copy to forward requests to.

```
$ more exp_25_rec.tcl
#Edit variables
for {set l 0} {$l < $anillos} {incr l} {
set downerlimit [expr $l * ($numnodes + 1)]
 for {set i $downerlimit} {$i<($starth + $downerlimit)} {incr i} {
 #puts "Daten setzen für Ressource $i"
  # Info resource
  ! n$i/resource setPrice 0 1 6
  ! n$i/resource setPrice 1 17 27
  ! n$i/resource setMoney_balance 20000

  #Info serviceCopy en puerto 103
  ! n$i/servicecopy103 setServiceID 2
  ! n$i/servicecopy103 setPrice 0 5 11
  ! n$i/servicecopy103 setPrice 1 25 35
  ! n$i/servicecopy103 setPrice 2 20 30
  ! n$i/servicecopy103 setPrice 3 20 30
  ! n$i/servicecopy103 setMoney_balance 20000
  ! n$i/servicecopy103 setMasterServiceCopy $numnodes 101
 }
}

#puts "Set variables in the outer leaves ..."

for {set l 0} {$l < $anillos} {incr l} {
set downerlimit [expr $starth + ($l * ($numnodes + 1))]
for {set i $downerlimit} {$i<($numnodes + ($l * ($numnodes + 1)))} {incr i} {
  #puts "Daten setzen für Client&SC $i"
  # Info resource
  ! n$i/resource setPrice 0 1 6
  ! n$i/resource setPrice 1 20 27
  ! n$i/resource setMoney_balance 10000

  #Info serviceCopy en puerto 103
  ! n$i/servicecopy103 setServiceID 2
  ! n$i/servicecopy103 setPrice 0 5 11
  ! n$i/servicecopy103 setPrice 1 17 35
  ! n$i/servicecopy103 setPrice 2 20 30
  ! n$i/servicecopy103 setPrice 3 20 30
  ! n$i/servicecopy103 setMoney_balance 10000
  ! n$i/servicecopy103 setMasterServiceCopy $numnodes 101

  # Client
    #set currressource [expr ($i - ($level1*$level2) + ($level1*$level3))/($level3)]
    set currressource [expr ($i - ($level1*$level2) - ($l*($numnodes + 1)) + ($level1*$level3))/($level3) + ($l *
($numnodes + 1))]
    ! n$i/client setKnownResources 1 [java::new long\[\] 1 [expr $currressource]]
    ! n$currressource/resource addKnownClient [expr $i] 102
    ! n$i/client setPrice 2 25 70
    ! n$i/client setPrice 3 23 51
    ! n$i/client setMoney_balance 40000
```

```
  }
}
```

Next, the "density level" is set up in *exp_25_rec_den2.tcl*. Like already mentioned above, all resource nodes could contain service copies/instances. However, the ratio of capacity of resources and the number of service copies is variable. Thus, these density levels differ in the capacity of the resources and the placement – respectively concentration – of the service copies. The number of deployable services is always fixed to 300 (resource capacity * number of service copies = const.) to guarantee comparability. For a high concentration scenario ("low density"), many copies are situated on a few resource nodes which have a high capacity of resources to offer. SCDis0.txt to SCDis4.txt determine the number of service copies and also their placement and are generated by
java catnet.util.createSCDistribution $number_of_rings_per_node $rings.
For density level 2, all existing resource nodes have 12 resource units to sell. SCDis2.txt contains 25 numbers of resource nodes, where the service copies should be placed on. exp_25_rec_den2.tcl reads the text file and instantiates the service copies. 25 service copies are set on 25 different resource nodes and each resource node can sell 12 resources. Please note, that resource nodes are on all nodes, but only a few of them host service copies.

```
$ more exp_25_rec_den2.tcl
set f [open "SCDis2.txt" "r"]
set out [read $f]
close $f

foreach i [split $out "\n"]  {
   if {$i > 0} {
   if {$i < ($numnodes+($anillos-1)*($numnodes+1))} {
      puts "Construct node $i for Density 2"
      set knowscs_ [java::new long\[\] 1]
      set knowscsID_ [java::new int\[\] 1]
      set knowscsPort_ [java::new int\[\] 1]
      $knowscs_ set 0 [expr $i]
      $knowscsID_ set 0 2
      $knowscsPort_ set 0 103
      ! n$i/resource setKnownServiceCopies 1 $knowscs_ $knowscsPort_ $knowscsID_
      ! n$i/resource setBW 12
      }
      }
}
```

exp_25_dem.tcl is the last script, that starts the simulation (the *rt* . commands). The *script {}* commands start some methods in the agents, like periodical services (to pay tax periodically to the system), initiate service requests on the clients (startDemand) etc. Do not change anything in this script.

```
$ more exp_25_dem.tcl
puts "Start NAM-Support"
#set nam [java::call drcl.inet.InetUtil setNamTraceOn [! .] "100.nam" [_to_string_array "red blue yellow green
black orange"]]
set complogfilename "./logs/$netlogfilename"
set nam [java::call catnet.util.CatNetMonitor setCatNetMonitorOn [! .] $complogfilename]

puts "Start simulation...
"
set time_ 0.001
set dynTop_ [mkdir catnet.util.DynamicTopology .dynTop]
set sim [attach_simulator .]
```

```
puts "Start demand"
script {! n*/client startDemand $demandfilename} -at 0.05 -on $sim
script {! n*/client start_periodical_services 5} -at 0.05 -on $sim
script {! n*/resource start_periodical_services 5} -at 1.05 -on $sim
script {! n*/servicecopy103 start_periodical_services 5} -at 0.05 -on $sim
script {! .dynTop setDynamicTopologyOn [! /test]} -at 0.02 -on $sim

rt . stop
run .
rt . resume
```

Launch of experiments

The experiments are launched with the shell script frun.sh. The compulsory command line arguments are described below.

```
$ ./frun.sh
Use: frun DemandQuantity DemandTime Rings Repetitions ExperimentName
```

DemandQuantity: Length of demand queue (randomly spread over the clients)
Values of 2.000-10.000 are reasonable

DemandTime: minimum time between requests, changes have a significant impact on the results (used spectrum: 25ms-100 ms)

Rings: this value describes the number of rings (each 106 nodes) used for the simulation topology (for CatNet we exclusively used 1 ring). See Figure 26.

Repetitions: number of repetitions the experiment shall be run (implemented for a statistical analysis, in CatNet always set to 1)

ExperimentName: every experiment should have a name, indicating the set of values or whatever might be reasonable. The log files will be stored in a directory "logs/ExperimentName". Thus, a simulation launch could look like this:

```
$ ./frun.sh 2000 75 1 1 test &
```

frun.sh distributes the jobs on the connected machines. These connected machines are in the Freiburg configuration a cluster of 16 double processors on 16 interconnected computers with one file system. The URLs of the machines are loginX.uni-freiburg.de. The X is filled from all numbers in *machinas.txt* (1 to 16), so login1.uni-freiburg.de is a valid example. This should be adjusted to the conditions on the platform the simulation hould run on. In *frun.sh* some while-loop distribute the jobs on the machines.

```
while [ "$i" -lt 5 ]
   do
     local j=0;
     while [ "$j" -lt 5 ]
     do
       (( k+= 1 ))
       echo "Executing ./exp25${i}${j}_${rings}.sh on login$(($k%16+1))"
       ssh "login$(($k%16+1))" "source /home/reinickm/catnet/setPath.sh;cd /home/reinickm/ catnet/script; nohup
./exp25${i}${j}_${rings}.sh &" &
       (( k+=1 ))
       echo "Executing ./exp25b${i}${j}_${rings}.sh on login$(($k%16+1))"
```

```
      ssh "login$(($k%16+1))" "source /home/reinickm/catnet/setPath.sh;cd /home/reinickm/ catnet/script; nohup
./exp25b${i}${j}_${rings}.sh &" &
       (( j+=1 ))
     done
     (( i+=1 ))
   Done
```

The outer while loop runs all dynamics scenarios, the inner one sets the density levels. In each loop circle, an ssh connection is built and the environment is set and the simulation run launched. It is recommendable to use the *nohup* command, to avoid loosing the processes when the connection fails during the simulation.

When the simulation finishes successfully, the evaluation is run immediately and an email is sent to *reinicke@uni-bayreuth.de*, containing an attachment with the aggregated results. This address can be adjusted in the last lines of *frun.sh*.

An experiment (including 50 scenarios), set up with a demand queue of 2000, may take up to 1 hour, dependent on the simulation platform.

./status.sh shows the number of jobs running on all machines

./stop.sh kills all experiment jobs on the machines

Evaluation of raw experimentation results
When the experiments are finished, the raw results are to be processed and be put together in one file. This can be initiated by the command *$ sh get-logs.sh*, which calls the script *juntar_logs.sh* and *juntar_logs.p*l. These scripts copy the logs to the directory ExperimentName and calculate the metrics response time (REST), service distance, allocation efficiency (RAE), social welfare (SWF), # of messages and # of message hops.

REST: How long does it take on average to fill a request; time between "cfp" and "accept"

RAE: The ratio of matched transactions divided by the number of all proposals: #accepts/ #proposals

SWF: The sum of all individual utilities / income from negotiations

All results are stored in *tot.log*, which is depicted below.
When using the *frun.sh* script, these scripts are initiated automatically when the simulations are completed.

The results are presented in two row sets. The upper row always describes the catallactic data, the lower one the baseline approach. This file can now be used for further analysis.

$ more tot.log

REST	**serv. dist.**	**RAE**	**SWF**	**#Mess**	**#Hops**
Den0 / Dyn0					
1042.1692	5.174074	81.0	3340.1395535658676	49311	76707
268.5367	4.3853106	91.0	54641.142169686835	48078	74648

Den0 / Dyn1
997.18097 5.120211 76.0 3948.859722135588 49171 76165
390.625 4.592033 75.0 38305.681918176255 47997 74683

Den0 / Dyn2
715.4252 4.8339624 81.0 3900.231710857463 51842 79296
462.54504 4.623123 68.0 31839.423511077934 48148 74869

Den0 / Dyn3
813.18726 5.180305 72.0 8544.793870284384 49337 76431
531.9764 4.1739864 61.0 25703.634225436028 47962 74408

Den0 / Dyn4
765.76373 4.8220067 63.0 16118.718105337906 48876 75600
638.0683 4.113852 54.0 22039.828603541722 48453 75055

Den1 / Dyn0
667.33295 4.728049 82.0 3858.5112218442823 52451 80454
259.52737 3.7743018 92.0 56438.31634161309 51674 81126

Den1 / Dyn1
715.4252 4.8339624 81.0 3900.231710857463 51842 79296
337.73126 3.4780362 79.0 41966.05086153074 52633 82755

Den1 / Dyn2
672.16583 4.8329015 79.0 4761.467196100835 51732 79186
437.71494 3.402715 68.0 31944.72857394768 52683 83176

Den1 / Dyn3
715.4252 4.8339624 81.0 3900.231710857463 51842 79296
544.1159 3.268166 59.0 24402.0124574632 52540 82852

Den1 / Dyn4
715.4252 4.8339624 81.0 3900.231710857463 51842 79296
731.1757 3.3995817 49.0 18852.7071224058 53162 83886

Den2 / Dyn0
620.0532 4.9480357 80.0 3498.508561091525 58153 87553
250.72044 3.0563536 93.0 57299.63702543375 60258 99858

Den2 / Dyn1
607.365 4.9642415 82.0 3822.3491869351874 57536 86424
356.75632 2.758988 77.0 39482.89167042516 61641 103365

Den2 / Dyn2
601.9464 4.991272 82.0 4576.027380070221 57300 86187
460.4977 2.7557251 67.0 31385.36216960587 62746 105632

Den2 / Dyn3
579.13293 4.8886075 81.0 7293.879719026896 57140 85962
631.06226 2.7641509 54.0 21514.17967967256 62791 106097

Den2 / Dyn4
542.0889 4.4946094 76.0 18892.570642088165 58375 88462
709.23615 2.6324434 50.0 19104.46955254291 62882 106558

Den3 / Dyn0
601.0859 4.673077 79.0 3752.893403877148 68964 99435
247.69482 2.429054 91.0 55632.901188618285 74875 128876

Den3 / Dyn1
555.3876 4.5776396 81.0 4176.954373544788 69239 99606
340.09933 2.0556293 77.0 40499.722998574536 79325 137893

Den3 / Dyn2
558.13214 4.6683292 82.0 4593.692632761087 68167 98126
453.9735 2.172897 66.0 30990.009391433272 80885 141984

Den3 / Dyn3
527.33417 4.634783 82.0 5891.628974576442 67939 97932
593.6347 2.1457565 55.0 22059.496031275015 82025 144862

Den3 / Dyn4
485.2979 4.130829 79.0 20571.84634816571 69438 100858
697.3492 2.1363637 50.0 18432.458012472485 82405 145985

Den4 / Dyn0
601.2161 4.700521 77.0 3918.727433016509 79104 109894
248.02135 2.1089888 91.0 54831.009274982425 87696 152331

Den4 / Dyn1
586.67773 4.8280253 80.0 4076.161422768566 77477 107538
329.532 1.9084967 78.0 41214.266800785925 96492 172058

Den4 / Dyn2
715.4252 4.8339624 81.0 3900.231710857463 51842 79296
479.45734 1.8743961 63.0 28371.55460683753 99435 179072

Den4/ Dyn3
516.5589 4.452381 81.0 7069.994795747933 78855 109592
510.11804 1.8853289 61.0 25753.40992741804 100072 181094

Den4 / Dyn4
505.87903 4.032258 76.0 19980.32083520734 79713 111970
704.9562 1.914405 49.0 17862.27122642783 100921 183593

Scripts
For completeness the scripts *frun.sh* and the evaluation scripts are presented.

```
$ more frun.sh
#!/bin/bash

createDemand() {
   echo "Creating Demand"
   java catnet.util.createRandomDemand exp25_demmand.txt $demandQuantity 1 30 105 $rings 106 $demand-
Time
}

createSC() {
   echo "Creating SC"
   java catnet.util.createSCDistribution 105 $rings
}

startExperiments() {
   local currentRun=1;
   while [ "$currentRun" -le "$numEx" ]
   do
    runExperiment $currentRun
```

```
     echo
     echo "Run finished. Now creating logfiles..."
     getLogs $currentRun
     (( currentRun+=1 ))
   done
}

runExperiment() {
   local currentRun=$1
   echo
   echo "Now Starting Run" $currentRun
   date
   echo "====================="
   local i=0;
   local k=-1;
   while [ "$i" -lt 5 ]
   do
     local j=0;
     while [ "$j" -lt 5 ]
     do
       (( k+= 1 ))
       echo "Executing ./exp25${i}${j}_${rings}.sh on login$(($k%16+1))"
       ssh "login$(($k%16+1))" "source /home/reinickm/catnet/setPath.sh;cd /home/reinickm/catnet/script; nohup
./exp25${i}${j}_${rings}.sh &" &
       (( k+=1 ))
       echo "Executing ./exp25b${i}${j}_${rings}.sh on login$(($k%16+1))"
       ssh "login$(($k%16+1))" "source /home/reinickm/catnet/setPath.sh;cd /home/reinickm/catnet/script; nohup
./exp25b${i}${j}_${rings}.sh &" &
       (( j+=1 ))
     done
     (( i+=1 ))
   done
   #list the jobs
   #we need to wait untill all jobs have finished
   echo "Waiting for run to complete"
   wait
   date
   echo "All jobs completed."
}


getLogs() {
  sh get-logs.sh ${experimentName}"/"$1
}

sendMail() {
   echo "Sending Notification"
   date
   echo "Experiment $experimentName finished." | mail -s "StatusReport: frun.sh :: $experimentName" re-
inicke@uni-bayreuth.de -a /home/reinickm/catnet/script/logs/$experimentName
/tot.log
}

#THE SCRIPT ACTUALLY STARTS HERE

if [ -z "$5" ]; then
   echo "Use: frun DemandQuantity DemandTime Rings Repetitions ExperimentName"
else
   #now rename input parameter variables
   demandQuantity=$1
   demandTime=$2
   rings=$3
```

```
    numEx=$4
    experimentName=$5
    echo "Codename: ${experimentName}. We will be doing ${numEx} Repetitions and use ${rings} Rings. Demand Quantity is ${demandQuantity}, Demand Time is ${demandTime}"
    echo "================================================================================================================="
    #create the sc distribution
    createSC
    #create a demand
    createDemand
    #start the java subprocesses
    startExperiments
    #process logfiles
    getLogs
    #send notify email
    sendMail
fi

$ more get-logs.sh
#!/bin/zsh
INDX=0
#LOGDIR=logs/logs_`date +'%Y%m%d_%H%M'`
LOGDIR=logs/$1
mkdir -p "$LOGDIR"
for DIN in 0 1 2 3 4
do
  for DEN in 0 1 2 3 4
  do
    #CURR=${MACHINAS[$INDX]}
    #MACHINA=arvei-"$CURR"
    LOGC=exp25"$DIN""$DEN"_graph.log
    LOGC_STUFF=exp25"$DIN""$DEN"_somestuff.log
    LOGB=exp25b"$DIN""$DEN"_graph.log
    LOGB_STUFF=exp25b"$DIN""$DEN"_somestuff.log
    echo "[** -- Computing exp25$DIN$DEN and exp25b$DIN$DEN **]"
    #scp "$MACHINA":"./catnet/script/logs/*.log" ./"$LOGDIR"
    #ssh "$MACHINA" "cd catnet/script; rm -f; logout;"
    #echo $LOGDIR/$LOGC $LOGC_STUFF
    java catnet.util.TimeComputeSomeStuff logs/"$LOGC" "$LOGDIR"/"$LOGC_STUFF"
    java catnet.util.TimeComputeSomeStuff logs/"$LOGB" "$LOGDIR"/"$LOGB_STUFF"
    cp logs/exp25"$DIN""$DEN".log "$LOGDIR"
    cp logs/exp25b"$DIN""$DEN".log "$LOGDIR"
    cp logs/exp25"$DIN""$DEN"_rae.log "$LOGDIR"
    cp logs/exp25b"$DIN""$DEN"_rae.log "$LOGDIR"
    cp logs/exp25"$DIN""$DEN"_netinfo.log "$LOGDIR"
    cp logs/exp25b"$DIN""$DEN"_netinfo.log "$LOGDIR"
  done
done
/home/reinickm/catnet/script/juntar_logs.sh "$LOGDIR" >"$LOGDIR/tot.log"
#tar -cf "$LOGDIR".tar.gz "$LOGDIR"

$ more juntar_logs.sh
cd $1
INDX=1
for DEN in 0 1 2 3 4
do
  for DIN in 0 1 2 3 4
  do
    LOGC=exp25"$DIN""$DEN".log
    LOGCW=exp25"$DIN""$DEN"_swf.log
    LOGCR=exp25"$DIN""$DEN"_rae.log
```

```
    LOGCS=exp25"$DIN""$DEN"_somestuff.log
    LOGCN=exp25"$DIN""$DEN"_netinfo.log
    LOGB=exp25b"$DIN""$DEN".log
    LOGBW=exp25b"$DIN""$DEN"_swf.log
    LOGBR=exp25b"$DIN""$DEN"_rae.log
    LOGBS=exp25b"$DIN""$DEN"_somestuff.log
    LOGBN=exp25b"$DIN""$DEN"_netinfo.log
    cat "$LOGC" | grep SWF | cut -f 2 >"$LOGCW"
    /home/reinickm/catnet/script/juntar_logs.pl $LOGCR $LOGCS $INDX $LOGCN $LOGCW
    cat "$LOGB" | grep SWF | cut -f 2 >"$LOGBW"
    /home/reinickm/catnet/script/juntar_logs.pl $LOGBR $LOGBS $INDX $LOGBN $LOGBW
    echo ""
  done
  echo ""
done
cd ..

$ more juntar_logs.pl
#!/usr/bin/perl

open(fitxer1,$ARGV[0]);
open(fitxer2,$ARGV[1]);
open(fitxer3, $ARGV[3]);
open(fitxer4, $ARGV[4]);

$line1 = <fitxer1>;
($tmpa, $tmpb, $RAE) = split(' ',$line1,3);
chop($RAE);
$RAE = substr($RAE,0,length($RAE)-1);

$ACCTIME = <fitxer2>;

$line3 = <fitxer3>;
($totalpackets, $totalhops) = split(' ', $line3, 2);
$totalpackets = substr($totalpackets,0,length($totalpackets)-1);
chop($totalhops);
$totalhops = substr($totalhops,0,length($totalhops)-1);

$SWF = <fitxer4>;
chop($SWF);

($info, $r) = split('_',$ARGV[0],2);
($r, $bits) = split('25',$info,2);
if(length($bits) == 2){
        $sim = "0";
}
else{
        $sim = "1";
}
$densidad = chop($bits);
$dinamismo = chop($bits);

print STDOUT $ACCTIME, "\t", $RAE, "\t", $SWF, "\t", $totalpackets ,"\t", $totalhops, "\r\n";
```

8 References

Abramson, D., R. Buyya, et al. (2002). "A Computational Economy for Grid Computing and its Implementation in the Nimrod- G Resource Broker." Future Generation Computer Systems **18**(8): 1061-1074.

Adar, E. and B. A. Huberman (2000). "Free Riding on Gnutella." First Monday **5**(10).

Ardaiz, O., F. Freitag, et al. (2002). CatNet - Catallactic Mechanisms for Service Control and Resource Allocation in Large Scale Application-Layer Networks. Proc. Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin.

Arndt, O., B. Freisleben, et al. (1999). Batch Queueing in the WINNER Resource Management System. Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada.

Bachmann, F., L. Bass, et al. (2004). Software Architecture Documentation in Practice: Documenting Architectural Layers. Pittsburgh, PA, Carnegie Mellon University.

Balakrishnan, H., M. F. Kaashoek, et al. (2003). "Looking Up Data In P2P Systems." Communications of the ACM **46**(2): 43-48.

Bichler, M. (2001). The Future of E-Commerce: multi-dimensional market mechanisms. New York, Cambridge University Press.

Brenner, T. (1996). Learning in a Repeated Decision Process: A Variation-Imitation-Decision Model. Papers on Economics & Evolution. Jena, Max-Planck-Institut für die Erforschung von Wirtschaftssystemen.

Brenner, T. (2002). "A Behavioural Learning Approach to the Dynamics of Prices." Computational Economics **19**: 67-94.

Burghes, D. and A. Graham (1980). Introduction to control theory including optimal control. West Sussex, Ellis Horwood.

Bussmann, S. and K. Schild (2000). Self-Organizing Manufacturing Control: An Industrial Application of Agent Technology. Proc. of the 4th International Conference on Multiagent Systems (ICMAS), Boston.

Buyya, R. (2002). Economic-based Distributed Resource Management and Scheduling for Grid Computing, Monash University, Melbourne, Australia. **Ph.D.**

Buyya, R., D. Abramson, et al. (2001). A Case for Economy Grid Architecture for Service-Oriented Grid Computing. 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), San Francisco.

Buyya, R., D. Abramson, et al. (2002). "Economic models for resource management and scheduling in Grid computing." Journal of Concurrency and Computation: Practice and Experience **14**(13-15): 1507-1542.

Casanova, H., G. Obertelli, et al. (2000). The AppLeS parameter sweep template: user-level middleware for the grid. ACM/IEEE Conference on Supercomputing, Dallas, TX, IEEE Computer Society.

Catnet Project (2003). Assessment Final Report. Barcelona.

Chandra, P., A. Fischer, et al. (2001). "Darwin: Customizable Resource Management for Value-Added Network Services." IEEE Network **15**(1): 22-35.

Chawathe, Y., S. Ratnasamy, et al. (2003). Making Gnutella-like P2P Systems Scalable. ACM SIGCOMM'03, Karlsruhe, Germany, ACM Press.

Cheng, J. Q. and M. P. Wellman (1998). "The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes." Computational Economics **12**: 1-24.

Chun, B., D. Culler, et al. (2003). "PlanetLab: An Overlay Testbed for Broad-Coverage Services." ACM Computer Communication Review **33**(3).

Chun, B. N. and D. E. Culler (2000). Market-based Proportional Resource Sharing for Clusters. Berkeley, University of California.

Clearwater, S. H. (1996). Market-based control. A paradigm for distributed resource allocation. Singapore, World Scientific.

Cliff, D. and J. Bruten (1998). Less than Human: Simple adaptive trading agents for CDA markets. Proceedings of the IFAC Symposium on Computation in Economics, Finance, and Engineering: Economic Systems (CEFES'98).

Cohen, B. (2003). Incentives build robustness in BitTorrent. Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA.

Erev, I. and A. E. Roth (1998). "Predicting How People Play Games - Reinforcement Learning in Experimental Games with Unique, Mixed Strategy Equilibria." American Economic Review **88**(4): 848-881.

Eymann, T. (2003). Digitale Geschäftsagenten. Heidelberg, Springer Xpert.press.

Eymann, T. and H. Morito (2004). Privacy Issues of Combining Ubiquitous Computing and Software Agent Technology in a Life-Critical Environment. Proc. of the IEEE International Conference on Systems, Man and Cybernetics, Den Haag, IEEE Press.

Eymann, T., B. Padovan, et al. (2002). "A Prototype for an Agent-based Secure Electronic Marketplace Including Reputation Tracking Mechanisms." International Journal of Electronic Commerce **6**(4): 93-114.

Fausett, L. V. (1994). Fundamentals of Neural Networks, Prentice Hall.

Ferguson, D. F., C. Nikolaou, et al. (1996). Economic Models for Allocating Resources in Computer Systems. Market-Based Control - A Paradigm for Distributed Resource Allocation. Singapore, World Scientific**:** 156-183.

Foster, I., C. Kesselman, et al. (1999). A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation. Proceedings of the International Workshop on Quality of Service.

Freedman, M. J., E. Freudenthal, et al. (2004). Democratizing Content Publication with Coral. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation, San Francisco, CA, ACM Press.

Frey, J., T. Tannenbaum, et al. (2002). "Condor-G: A Computation Management Agent for Multi-Institutional Grids." Cluster Computing **5**(3): 237-246.

Friedman, D. P. (1993). The Double Auction Market. Santa Fe Institute studies in sciences of complexity. Reading, Massachussets., Addison-Wesley.

Furubotn, E. G. and R. Richter (1998). Institutions and Economic Theory: The Contribution of the New Institutional Economics. Detroit, University of Michigan Press.

Gehring, J. and A. Reinefeld (1996). " MARS - A Framework for Minimizing the Job Execution Time in a Metacomputing Environment." Future Generation Computer Systems **12**(1): 87-99.

Goldberg, D. (1993). Genetic Algorithms in Search, Optimization and Machine Learning. Reading MA, Addison Wesley.

Gomoluch, J. and M. Schroeder (2003). Market-based Resource Allocation for Grid Computing: A Model and Simulation. 1st International Workshop on Middleware for Grid Computing, Rio de Janeiro, Brazil.

Grimshaw, A. S., W. A. Wulf, et al. (1994). Legion: The Next Logical Step Toward a Nationwide Virtual Computer, University of Virginia.

Hayek, F. A. v., W. W. Bartley, et al. (1989). The collected works of F.A. Hayek. Chicago, University of Chicago Press.

Holland, J. H. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. Cambridge, MIT Press.

Hoppmann, E. (1999). Unwissenheit, Wirtschaftsordnung und Staatsgewalt. Freiheit, Wettbewerb und Wirtschaftsordnung. V. Vanberg. Freiburg, Haufe Verlag**:** 135-169.

Horwitz, S. (2003). Catallaxy, Competition, and 21st Century Capitalism: An Agenda for Economics. Annual Conference of the International Confederation for the Advancement of Pluralism in Economics, Kansas City, University of Michigan Press.

Huberman, B. A. (1988). The Ecology of Computation. Amsterdam, North-Holland.

IBM (2/12/01/). Autonomic Computing Manifesto. Yorktown Heights, NY, IBM.

Izal, M., G. Urvoy-Keller, et al. (2004). Dissecting BitTorrent: Five Months in a Torrent's Lifetime. Passive and Active Measurements (PAM), Antibes Juan-les-Pins.

Jennings, N. R., P. Faratin, et al. (2001). "Automated negotiation: prospects, methods and challenges." International Journal of Group Decision and Negotiation **10**(2).

Kearney, P. J., R. E. Smith, et al. (2000). "Integration of computational models inspired by economics and genetics." BT Technology Journal **18**(4): 150-161.

Kephart, J. O. and D. Chess (2003). "A Vision of Autonomic Computing." IEEE Computer **36** (1): 41-50.

Kephart, J. O., J. E. Hanson, et al. (1998). Dynamics of an information filtering economy. Second International Workshop on Cooperative Information Agents (CIA'98), Heidelberg, Springer.

Krauter, K., R. Buyya, et al. (2001). "A taxonomy and survey of grid resource management systems for distributed computing." Software - Practice And Experience **32**(2): 135-164.

Lavoie, D., H. Baetjer, et al. (1990). "High-Tech Hayekians - Some Possible Research Topics in the Economics of Computation." Market Process **8**(Spring): 120-147.

Lomuscio, A. R., M. J. Wooldridge, et al. (2000). A classification scheme for negotiation in electronic commerce. LNAI. Heidelberg, Springer**:** 19-33.

Malkhi, D., M. Naor, et al. (2002). Viceroy: A scalable and dynamic emulation of the butterfly. 21st annual ACM symposium on Principles of distributed computing, ACM Press.

Miller, M. S. and K. E. Drexler (1988). Markets and Computation: Agoric Open Systems. The Ecology of Computation. B. A. Huberman. Amsterdam, North Holland**:** 133-176.

Mojo Nation. (2003). "Mojo Nation Website." Retrieved 2003/01/01/, from http://www.mojonation.net/.

Müller, J. and T. Eymann (2003). Optimizing Strategy in Agent-based Automated Negotiation. Wirtschaftsinformatik 2003, Dresden, Physica Verlag.

Nabrzyski, J., J. M. Schopf, et al., Eds. (2003). Grid Resource Management. Amsterdam, Kluwer.

Neal, R. M. (1996). "Bayesian Learning for Neural Networks." Lecture Notes in Statistics **118**.

North, D. C., R. Calvert, et al. (1990). Institutions, Institutional Change and Economic Performance - Political Economy of Institutions and Decisions. Cambridge, Cambridge University Press.

Preist, C. (1998). Economic Agents for Automated Trading. Bristol, Hewlett Packard Laboratories.

Press, W. H., S. A. Teukolsky, et al. (2002). Numerical Recipes in C++ - The Art of Scientific Computing. Cambridge, MA, Cambridge University Press.

Pruitt, D. G. (1981). Negotiation behavior. New York, Academic Press.

Rabinovich, M. and A. Aggarwal (1999). RaDaR: A scalable architecture for a global Web hosting service. The 8th International World Wide Web Conference, Toronto.

Ratnasamy, S., P. Francis, et al. (2001). A Scalable Content-Addressable Network. Conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, ACM Press.

Regev, O. and N. Nisan (1998). The POPCORN market - an online market for computational resources. International Conference on Information and Computation Economies, Charleston, SC, ACM Press.

Ripeanu, M. (2001). Peer-to-Peer Architecture Case Study: Gnutella Network. Chicago, University of Chicago.

Ripeanu, M., I. Foster, et al. (2002). "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design." IEEE Internet Computing **6**(1).

Rosenschein, J. S. and G. Zlotkin (1994). Rules of encounter - designing conventions for automated negotiation among computers. Cambridge, MIT Press.

Rowstron, A. and P. Druschel (2001). Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. 18th IFIP/ACM International Conference on Distributed Systems Platforms, Springer.

Roy, A. and V. Sander (2003). GARA: A Uniform Quality of Service Architecture. Grid Resource Management: State of the Art and Future Trends. J. Nabrzyski, J. M. Schopf and J. Weglarz. Amsterdam, Kluwer Academic Publishers**:** 377-394.

Sandholm, T. W. (1996). Negotiation Among Self-Interested Computationally Limited Agents. Amherst, University of Massachusetts.

Sandholm, T. W. and R. H. Crites (1995). On Multiagent Q-Learning in a Semi-competitive Domain. IJCAI-95 Workshop on Adaptation and Learning in Multi-agent Systems.

Saroiu, S., P. K. Gummadi, et al. (2002). A measurement study of peer-to-peer file sharing systems. SPIE/ACM Conference on Multimedia Computing and Networking (MMCN 2002).

Sarshar, N., P. O. Boykin, et al. (2004). Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable. Fourth International Conference on Peer-to-Peer Computing (P2P'04), Zürich, Switzerland, IEEE Computer Society.

Sathi, A., M. S. Fox, et al. (1989). Constraint-directed Negotiation of Resource Allocations. Distributed Artificial Intelligence 2, Morgan Kaufmann.

Simon, H. A. (1957). Models of Man - Social and Rational. New York, John Wiley & Sons.

Singh, M. P. and M. N. Huhns (2005). Service-Oriented Computing: Semantics, Processes, Agents. New Jersey, John Wiley & Sons.

Smale, S. (1976). "Dynamics in general equilibrium the theory." American Economic Review **66**(2): 284-294.

Smith, A. (1776). An inquiry into the nature and causes of the wealth of nations. London, Printed for W. Strahan; and T. Cadell.

Smith, R. E. and N. Taylor (1998). A Framework for Evolutionary Computation in Agent-Based Systems. Proceedings of the 1998 International Conference on Intelligent Systems, ISCA Press.

Snelling, D. and T. Priol (2003). Next Generation Grid(s). European Grid Research 2005 - 2010. Brussels, Information Society - DG, Grids for Complex Problem Solving.

Stonebraker, M., R. Devine, et al. (1994). An economic paradigm for query processing and data migration in Mariposa. 3rd International Conference on Parallel and Distributed Information Systems, Austin, TX.

Sycara, K., L. Gasser, et al. (1989). Multi-agent Compromise via Negotiation. Distributed Artificial Intelligence 2, Morgan Kaufmann.

T-Online AG (2003). T-Online PDF Creator: Easy document conversion.

Tanenbaum, A. S. (1996). Computer Networks. Upper Saddle River, N.J, Prentice Hall PTR.

Tesfatsion, L. (1997). How Economists can get Alife. Santa Fe Institute Studies. W. B. Arthur, S. Durlauf and D. A. Lane. Redwood City, CA, Addison Wesley**:** 533-564.

Truex, D. P., R. Baskerville, et al. (1999). "Growing Systems in Emergent Organizations." Communications of the ACM **42**(8): 117-123.

Varian, H. R. (1995). Mechanism Design for Computerized Agents. Talk presented at the Usenix Workshop on Electronic Commerce, New York.

Waldspurger, C. A., T. Hogg, et al. (1992). "Spawn: A Distributed Computational Economy." IEEE Transactions on Software Engineering **18**(2): 103-117.

Walras, L. (1954). Elements of pure economics. London, Allen and Unwin.

Weinhardt, C., C. Holtmann, et al. (2003). "Market Engineering." Wirtschaftsinformatik **45**(6): 635-640.

Weiser, M. (1991). "The Computer for the Twenty-First Century." Scientific American **265**(3): 94-104.

Wellman, M. P. (1993). "A market-oriented programming environment and its application to distributed multicommodity flow problems." Journal of Artificial Intelligence Research **1**: 1-23.

Wellman, M. P. (1996). Market-Oriented Programming: Some Early Lessons. Market-Based Control: A Paradigm for Distributed Resource Allocation. S. H. Clearwater. Singapore, World Scientific**:** 74-95.

Wellman, M. P. and W. E. Walsh (2003). "Decentralized supply chain formation: A market protocol and competitive equilibrium analysis." Journal of Artificial Intelligence Research **19**: 513-567.

Wolski, R., J. Brevik, et al. (2003). Grid Resource Allocation and Control Using Computational Economies. Grid Computing: Making The Global Infrastructure a Reality. F. Berman, G. Fox and A. Hey. San Francisco, Wiley & Sons.

Wooldridge, M. J. (1999). Intelligent Agents. Multiagent Systems. G. Weiss. Cambridge, MA, MIT Press**:** 27-78.

Yamaki, H., M. P. Wellman, et al. (1996). A Market-Based Approach to Allocating QoS for Multimedia Applications. Second International Conference on Multiagent Systems (ICMAS'96), Kyoto.

Ygge, F. (1998). Market-Oriented Programming and its Application to Power Load Management, Lund University, Sweden. **Ph.D.**

Zhao, B., J. Kubiatowicz, et al. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Berkeley, Computer Science Division, University of California.

Die Arbeit beschreibt die Spezifikation und das Design von Softwarekomponenten, um das Konzept der Katallaxie in Grid-Systemen umzusetzen. Eine Einführung ordnet das Konzept der Katallaxie in bestehende Grid-Taxonomien ein und stellt grundlegende Komponenten vor. Anschließend werden diese Komponenten auf ihre Anwendbarkeit in bestehenden Application Layer Netzwerken untersucht.

ISSN 1864-9300