CatNet

ITS-2001-34030

Evaluation of the Catallaxy paradigm for decentralized operation of dynamic application networks

Delivery 1: Simulator

Covering period 1.3.2002-30.8.2002

Report Version: 0.1

Report Preparation Date: 01.09.02

Classification: public

Contract Start Date: 01.03.2003 Duration: 12 months

Project Co-ordinator: Universitat Politècnica de Catalunya

Partners: Albert-Ludwigs-Universität Freiburg

Project funded by the European Community under the "Information Society Technologies" Programme (1998-2002)

Table of contents:

DELIVERABLE SUMMARY SHEET

Project Number: ITS-2001-34030

Project Acronym: CatNet

Title: Evaluation of the Catallaxy paradigm for decentralized operation of dynamic application networks

Deliverable N°: 1 - Simulator

Due date: 08/2002

Delivery Date: 09/2002

Short Description:

We describe a simulator for a generic application layer network (ALN). This ALN simulator is implemented on top of the JavaSim 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 such as service access, bandwidth and storage are coordinated in the network.

The CATNET simulator employs two main control mechanisms for network coordination: a "baseline" control mechanism and a "catallactic" control mechanism. The baseline control mechanism computes the resource allocation decision in a centralized service/resource provider. The catallactic control mechanism has the characteristic that its resource allocation decisions are carried out by self-interested agents with only local information about the environment. Each agent has a resource discovery facility and a negotiation strategy module.

The trace collection of the simulation execution is done via log files for processing at a later stage after the simulation.

Partners owning: Universitat Politècnica de Catalunya

Partners contributed: Albert-Ludwigs-Universität Freiburg

Made available to: Public

1. Introduction

This document describes the development of the CatNet simulator for application layer networks (ALN). Application layer networks such as peer-to-peer networks and content distribution networks obtain increased interest from the research community. Grid technology for distributed processing is another topic closely related to this area. The intention of the CatNet simulator is to be a simulator for such application layer networks with the possibility to evaluate different resource allocation resp. network control strategies.

The purpose of application layer networks is to provide a infrastructure for distributed demand and supply of data or computation services for applications. The provisioning of these services is constrained by the scarcity of network resources, e.g. storage, bandwidth, or CPU cycles. To provide an optimal service allocation on the application layer, the deployment and allocation of the network resources has to be optimised. Optimization criterions can be based on economic cost efficiency, technical performance measures or a combination of these. In this deliverable, our goal is to develop a simulator, which allows to experimentally compare two main resource allocation strategies: A centralized approach in which decisions are taken centrally and a decentralized approach, where local agent negotiate resources using economic models.

2. Specification and design consideration

2.1 Selection of the network simulator

The design of an simulator for the application layer of a computer network bases on the availability of an implementation of the lower layer's communication protocols. There are a number of network simulators, which could be used as basis for the CatNet simulator. We evaluated two known network simulators to decide on top of which one the CatNet simulator will be implemented.

2.1.1 JavaSim

JavaSim is a component-based, compositional simulation environment [2]. It has been built upon the notion of the autonomous component programming model. Similar to COM/COM+, JavaBeans™, or CORBA, the basic entity in JavaSim is components, but unlike the other component-based software packages/standards, components in JavaSim are autonomous.

For the purpose of network modelling and simulation, Javasim defines on top of the autonomous component architecture a generalized packet switched network model. The model defines the generic structure of a node (either an end host or a router) and the generic network components, both can then be used as base classes to implement protocols across various layers.

JavaSim is developed and maintained by a research group of the State University of Ohio.

One of the important characteristics of JavaSim is that it has been developed entirely in Java. This feature makes reuse of earlier developed code of the project partners easy.

2.1.2 ns-2 network simulator

Ns-2 is a discrete event simulator targeted at networking research [4]. Ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. It is a widely used tool for research in networks.

The main drawback we found by our own experimental work is the scalability of ns-2, which can lead to memory problems when simulating large network topologies, as in the CatNet project.

2.1.3 Choice of Javasim

We have identified the following main advantages of JavaSim over ns-2:

1. Performance. In [2] JavaSim is compared with ns-2. It is shown that especially for large network topologies the performance of JavaSim is more robust than ns-2.

2. Portability. JavaSim has been developed entirely in Java. This makes JavaSim a truly platform-neutral environment.

3. Reusability. Java code developed in this project could be reused in middleware for real distributed applications.

We have implemented the CatNet simulator of the application layer network on top of the JavaSim network simulator.

2.2 Definitions of terms

Client: a computer program on a certain host, which needs access to a network service to fulfil its design objectives. The Client (C) tries to access that "service" at an arbitrary location within the computer network, use it for a defined time period, and then continues with its own program sequence. Client programs run on a connected network "resource". An example for a client would be a word processor program, which needs to generate an PDF-version of a text. In this case, the client goes through the following sequence. It (1) searches a "PDF conversion"-service in the network, then (2) negotiates for its use, (3) sends the original text, (4) receives the PDF version, (5) pays for service usage and (6) continues with whatever the user commands.

Service: an instantiation of an abstract and general application function, embodied in a computer program. An example would be the "PDF conversion" service as a computer program provided by the original manufacturer or a third party. A client that wants to access a service needs in fact to address one instance of the service, a service copy.

Service Copy: one instance of the "service". The service copy (SC) is hosted on a "resource" computer, which provides both storage space and bandwidth for the access of the service.

Resource: a host computer, which provides a limited number of storage space and access bandwidth for computer programs. Resources (R) are connected to each other via dedicated network connections. For simplicity in simulation, the CatNet resources provide both storage and bandwidth in an equal number of "service slots". Each service slot provides reserved space of both resource types for one service copy.

Network Connections: These connections are intended to be of equal length and thus of equal transmission time and costs. In the simulator they are set up in a script which builds up the simulation scenario.

2.3 The Application Scenario and Market Model

As an application scenario example we consider the distributed provisioning of Adobe's Acrobat (for creating PDF files) as a web service in an Akamai-like application layer network (data grid). Here, word-processor client programs would address the nearest/cheapest Acrobat service instance in order to create PDF files. The overall objective in the network would be (a) to always provide access to Acrobat service, such that a minimum number of service demands has to be rejected, and (b) to optimize network parameters such as provisioning costs and network communication.

2.3.1 The Dynamic Process of the service market

For accessing a service, the client needs to address a specific service_copy, allocate it for the time period needed and release it again afterwards. Which service copy the client addresses and allocates is a two-step process.

2.3.1.1 Addressing the service_copy

One mechanism variant to address a service_copy is a Gnutella-like routing algorithm. The client sends out a Call-for-proposal (cfp) message on its network connections to the next resource host. This host either provides a service copy of the requested type or not. If a service copy is available, the resource routes the request to the service copy (SC). The SC calculates an access price and sends a propose message back. If no service copy is available, the resource routes the cfp message further on to the next connected host, increasing the message hop counter by one. If the hop counter exceeds a given number, the message is discarded. The emitting client employs a time limit after which it decides to access one service copy out of the received answers.

The second mechanism employs a dedicated service coordinator ("master service copy", MSC), which is known by the SC of that service type. The cfp message is sent by the clients, who do not know about the infrastructure, again via the resources to any SC. The SC then relays the cfp to the MSC. The MSC knows where all service copies are held and their allocation schemes. In addition, it is able to compute the costs of providing a service and sends back a propose message revealing the "cheapest" SC to the client. The client will then "choose" the best SC from this one-item list.

2.3.1.2 Allocating the service_copy

If the client has decided which service_copy to access, it will send an accept message to that SC for allocating some timeslot in the future. The SC holds a timetable containing the information about when it is already allocated and it will only negotiate for any empty timeslots. Contracts have to be fulfilled; a re-negotiation of allocations is out of the project's scope.

2.3.2 Negotiation and dynamic prices

Starting with demand and supply prices given in an input data script, the prices will change according to the following scheme: Clients will always choose any cheapest available service copy from the list constructed by received propose messages. A reservation price does not exist. Service_copies will lower their initial offer price by 1 money unit if having send a propose message back and not having received an accept message in return. They will raise their initial offer price by 1 money unit after an offer has been accepted.
Resources will charge a fixed fee for bandwidth and storage to the SC price for the client. This price is increased by 1 for every resource (hops) in the connection chain, so that more distant SCs cost more than nearer ones even if their access price is the same.

2.3.3 Relocating service copies in the storage market

If a SC has been turned down several times (having answered with propose messages but never received an accept), demand and offer are no longer in equilibrium and the situation yields no utility gain for the SC anymore. In this case, the SC will be actively or passively relocate to another resource host.
In the decentralized approach, according to the majority of received messages, measured by incoming connection, the service_copy will send a cfp to the connected resource host and ask for a free slot. If that target resource is fully occupied, the SC will ask the second-often relay of request messages and so on. In the end, the SC opens up a new instance at the target resource host and deletes the old instance. The overall effect is that SCs move (themselves) around the network in the physical direction of the demand.

In a centralized approach, a "resource network coordinator" (RNC) exists. The SC wanting to relocate sends a cfp to that coordinator (who may also be a "master resource”), and it will be answered by a propose message which tells where to establish a new SC.

The costs of the relocating process of a SC, however, are not regarded in the project’s economic model for service access or resource usage.

2.3.4 Negotiation protocol

The objects of the CatNet simulator communicate by message flows concerning addressing, negotiation and fulfilment resp. money flows. This section outlines the communication protocols implemented between Clients, Service Copies, and Resources.

2.3.4.1 Message Flow

We consider two independent markets: The market for access of the application service (“service-market”) and the market for resource slots, representing storage and bandwidth (“storage-market”).

The messaging protocol to contemplate the storage market proceeds as follows. The Service_Copy calls for a proposal for storing itself on the resource. After a bargaining procedure (*call for proposal, reject, proposal, accept, confirm)* the Service_Copy will be hosted by the Resource (or not).

In the service market, the Resource will now be able to bargain with potential clients. After having reveived a *confirm/accept* by the client the Service_Copy approves service transmitting to the resource agent (*approve_service_providing*) and the transmission may begin (*provide_service*). Figure 1 shows this behavior. Money Flows shall take place after each transaction or after each period of time, respectively. Tracing of messages shall take place after each message. Both are not shown in the figure for the sake of clarity.

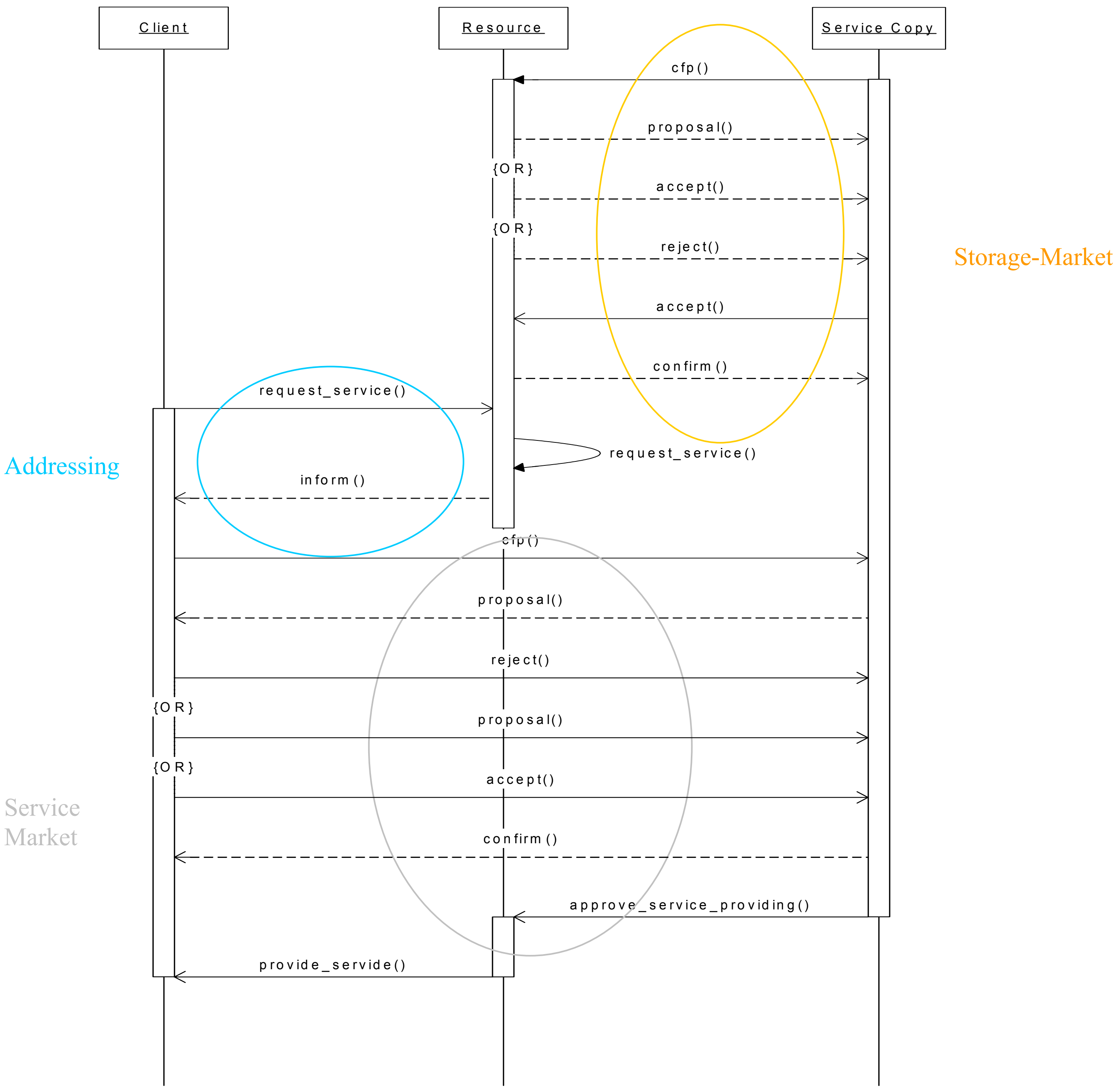

Figure 1. Sequence diagram of the communication and negotiation.

2.3.4.2 Money Flow

As resource costs are determining the money flows and several transactions may occur in one single period of time, service costs are accounted per transaction and storage costs per time span. Therefore, money flows in the decentralized approach can be separated in *flow per time* and *flow per transaction*.

The following flows can be identified:

–per time:

Service_Copy → Resource	(for usage of the storage)
Service_Copy → Service	(giving profit to principal)

–per transaction:

Client → Service_Copy	(for getting the service)
Service_Copy → Resource	(for usage of bandwidth)

2.4 Trace Collection

In order to evaluate the data after the simulation and to reconstruct the network configuration, we can both direct the log traces in a file and in a database. In our recent implementation, MySQL is used for logging because of the simplicity of usage, storage and query preparation. In addition, log files are written which allow to process the simulation data in posterior trace analysis.

After launching the simulation, the network configuration is stored into the database and during the bargaining phase additional data is sent to the LOG-Table. Figure 2 shows such the data structure of the log table, which also shows the relations between the several objects in the simulation. The central entity is the LOG-Table, which is joined to other tables. Client2Resource and Resource2Service describe the network and the allocation of services to resources. Relevant data can easily be transmitted, extracted and automatically be analyzed.

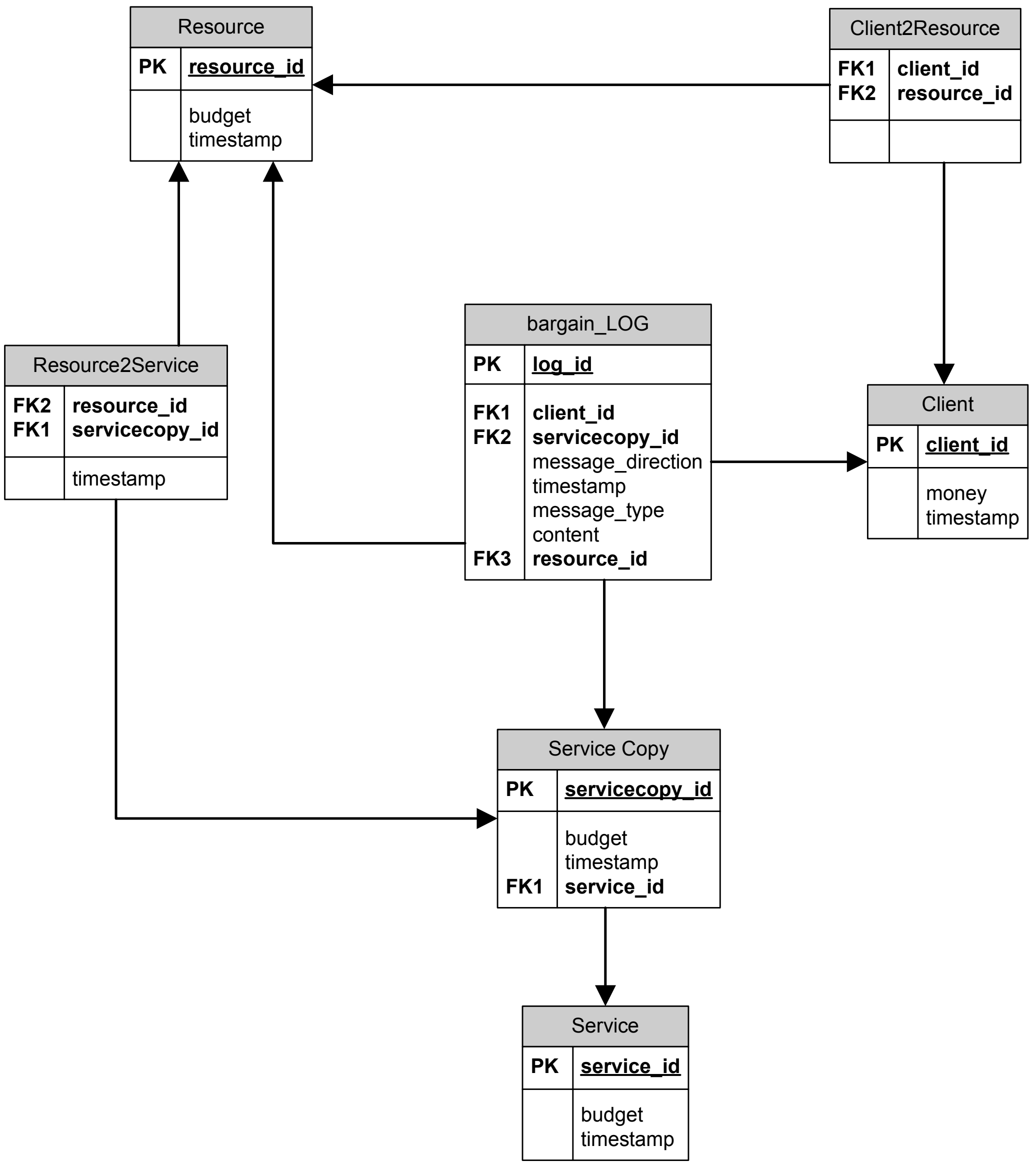

Figure 2. Model for the trace collection in the CatNet simulator.

3. Implementation

The CatNet application layer simulator is implemented on top of the JavaSim network simulator. Both simulators are programmed in Java. The setup of the simulation is done via a Tcl input data file. An example is given in annex 1.

We used JavaDoc for documenting the classes implemented in the CatNet simulator. The documentation on the Java classes and the source code of the simulator is available on http://research.ac.upc.es/catnet.

Figure 3 provides a summary of the packages and class hierarchy in the CatNet simulator.

In the CatNet simulator, one of the main classes to instantiate agents is the Agent_Source class. From this class, the service_copy, client, and resource agents are derived.

Each agent contains an instance of a strategy class, which governs the negotiation between the agents.

The class Agent_Source handles the communication between the agents and contains the methods to interpret the different messages received during the simulation.

In the following the hierarchy for the packages and classes in the CatNet simulator is shown:

Package Hierarchies:

catnet.accounting, catnet.agent, catnet.client, catnet.data, catnet.learning, catnet.learning.stdea, catnet.negotiation, catnet.resource, catnet.servicecopy, catnet.strategy, catnet.util, edu.uah.math.distributions

---

3.1 Class Hierarchy

- class java.lang.Object
  - class java.util.AbstractCollection (implements java.util.Collection)
    - class java.util.AbstractList (implements java.util.List)
      - class java.util.ArrayList (implements java.lang.Cloneable, java.util.List, java.util.RandomAccess, java.io.Serializable)
        - class catnet.util.**RingArray** (implements java.lang.Cloneable)
      - class java.util.Vector (implements java.lang.Cloneable, java.util.List, java.util.RandomAccess, java.io.Serializable)
        - class catnet.learning.stdea.**Genotype** (implements java.lang.Cloneable, catnet.learning.stdea.IMutateable, catnet.learning.stdea.IRandomizeable, java.io.Serializable)
  - class catnet.strategy.**avalanche** (implements catnet.strategy.IPricing)
  - class catnet.data.**constant**
  - class catnet.negotiation.**Conversation**
  - class edu.uah.math.distributions.**Data**
  - class catnet.data.**Debug**
  - class edu.uah.math.distributions.**Distribution**
    - class edu.uah.math.distributions.**ContinuousUniformDistribution**
    - class edu.uah.math.distributions.**Convolution**
    - class edu.uah.math.distributions.**NormalDistribution**
    - class edu.uah.math.distributions.**PoissonDistribution**
  - class edu.uah.math.distributions.**Domain**
  - class drcl.DrclObj (implements drcl.ObjectDuplicable, java.io.Serializable)
    - class drcl.comp.Component
      - class drcl.net.Module
        - class drcl.inet.application.SUDPApplication
          - class catnet.agent.**Agent_Source**
            - class catnet.client.**Client**
            - class catnet.resource.**Resource**
            - class catnet.servicecopy.**ServiceCopy**
          - class catnet.resource.**Baseline**
  - class catnet.accounting.**Factor** (implements java.lang.Comparable)
    - class catnet.negotiation.**ProposalLineItem**
    - class catnet.accounting.**StockLineItem**

- class catnet.learning.stdea.**Gene** (implements java.lang.Cloneable, catnet.learning.stdea.IMutateable, catnet.learning.stdea.IRandomizeable, java.io.Serializable)
    - class catnet.learning.stdea.**BooleanGene**
    - class catnet.learning.stdea.**FloatGene**
- class catnet.util.**Hp**
- class catnet.util.**IntCounter**
- class edu.uah.math.distributions.**IntervalData**
- class catnet.data.**Msg**
- class catnet.negotiation.**NegotiationHistory**
- class catnet.negotiation.**OrderbookItem** (implements java.lang.Comparable)
- class catnet.data.**pkt**
- class catnet.learning.stdea.**Plumage** (implements java.io.Serializable)
- class catnet.negotiation.**PriceDistribution**
- class catnet.negotiation.**Proposal**
- class edu.uah.math.distributions.**RandomVariable**
- class catnet.data.**resourceData**
- class catnet.learning.stdea.**Smith98** (implements catnet.learning.ILearning)
- class catnet.util.**TM_EVT**
- class catnet.util.**Tool**
- class catnet.strategy.**zeroIntelligence** (implements catnet.strategy.IPricing)

3.2 Interface Hierarchy

- interface catnet.learning.**ILearning**
- interface catnet.learning.stdea.**IMutateable**
- interface catnet.strategy.**IPricing**
- interface catnet.learning.stdea.**IRandomizeable**

Figure 3. Class and package hierarchy of the Catnet simulator.

In annex 2, a summary of the methods implemented in the main classes of the CatNet simulator is given, for details see the JavaDoc in http://research.ac.upc.es/catnet.

The main classes of the CatNet simulator are:

Class Agent_Source: Implements the communication between agents.
Class Client: Provides an agent representing a client.
Class Service_Copy: Provides an agent representing a service.
Class Resource: Provides an agent representing a resource.
Class Avalanche: Strategy class for negotiation between agents.

4. Preliminary results

Currently, the simulation of small size application networks consisting of clients, service_copies, and resource agents can be executed (see for instance the Tcl script in annex 1). The agents have been implemented with a negotiation protocol to trade resources.

The complete validation and debugging of the implementation with controlled experiments has shown to be complex due to the high number of messages in the negotiation protocol, and the broadcasting of messages. This observation leads to a continuing task for the next phase.

5. Discussion

In this section, we comment some of the observations made during the development of the simulator.

5.1 Observations on the development of the tasks of WP1

5.1.1 Development of a generic application network simulator

This work phase has started with the choice of which publicly available network simulator could be used as the base for the application layer simulator. We have chosen the JavaSim simulator programmed in Java. One of the reasons was the ease of reusing earlier code programmed in Java by the project partners.

During the development of a generic application layer simulator, we found that there is a trade-off between the programming complexity when complete generic usage is desired and the specific application scenarios we required. Currently, we focus on the simulation of the proposed application scenario. Other scenarios might need extending the simulator.

5.1.2 Configuring the control mechanisms

In order to ease performance comparison of different control mechanisms, our approach is to configure both the scenarios and the control mechanism through setup files to be read when the simulation starts. These files are Tcl files, which contain the definition of the network topology, the number of agents, initial prices and resources, etc. The Java classes remain unchanged for both control mechanisms.

Using configuration files, we expect that the performance results can be compared more easily. On the other hand, the simulator has to have sufficient flexibility to allow simulating the proposed environments. We observed that money flows may need to be introduced in the centralized mechanism in order to allow the comparison of the different resource allocation approaches.

We observed that the complete validation and debugging of the implementation is complex, especially when switching to simulations of larger topologies.

5.1.3 Trace collection

The current simulator allows collecting simulation data to be stored in log files. Work in progress focuses on implementing the storage of the simulation data in a database. The performance is currently quantified with money balance results of the agents.

5.2 Future work to be addressed

- The current simulator implementation has to be extended and validated on large-scale topologies.
- Exception handling and time-out implementation need to be improved.
- Dynamic network behavior has to be validated.
- Evaluation metrics have to be enhanced.

6. Conclusions

The CatNet simulator for application layer networks has been developed. Work, however, has to continue to adapt to the exact requirements of the simulations. The protocol for the communication between the agents contains a negotiation strategy, such that agents can trade resources. Our approach is to use configuration files to determine particular scenarios and to simulate different resource allocation strategies.

References:

[1] Oscar Ardaiz, Felix Freitag, Leandro Navarro, Torsten Eymann, Michael Reinicke. CatNet - Catallactic Mechanisms for Service Control and Resource Allocation in Large Scale Application-Layer Networks. *Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 21 - 24 May 2002, Berlin, Germany.
[2] http://www.javasim.org
[3] http://research.ac.upc.es/catnet
[4] http://www.isi.edu/nsnam/ns/

Annex 1

Example of a Tcl script for the simulation of a 10 node application network in the CatNet simulator

```
# 10 objects ...

puts "Constructing a network of 10 nodes"

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" "5orange3" "NeuerTest3"

puts "Create topology..."

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

# adjacency matrix
# (0)--(1)--(2)         0:resource, 1:router, 2:resource, 3,8,9,10:servicecopy
#       |`-(3)
#       `-(4)  4:client 5,6, resource, 7: client
set adjMatrix_ [java::new int\[\]\[\] 11 {{1} {0 2 3 4 5 6 7 8 9 10} {1} {1} {1} {1} {1} {1} {1} {1} {1}}]
java::call drcl.inet.InetUtil createTopology [! .] $adjMatrix_ $link_

puts "Building..."
set nb_ [mkdir drcl.inet.NodeBuilder .nb]
$nb_ setBandwidth 10000000; # 10Mbps

$nb_ build [! n?]
# client
$nb_ build [! h7] {
        udp                     drcl.inet.transport.UDP
        client  101/udp     catnet.client.Client
}

# n1 is a router

# resource
$nb_ build [! h2] {
        udp                     drcl.inet.transport.UDP
        resource  101/udp   catnet.resource.Resource
}
# resource
$nb_ build [! h5] {
        udp                     drcl.inet.transport.UDP
        resource  101/udp   catnet.resource.Resource
}
# resource
$nb_ build [! h6] {
        udp                     drcl.inet.transport.UDP
        resource  101/udp   catnet.resource.Resource
}
# resource
$nb_ build [! h0] {
        udp                     drcl.inet.transport.UDP
        resource  101/udp   catnet.resource.Resource
}
# servicecopy
$nb_ build [! h3] {
        udp                     drcl.inet.transport.UDP
        servicecopy 101/udp         catnet.servicecopy.ServiceCopy
}
# client
$nb_ build [! h4] {
        udp                     drcl.inet.transport.UDP
        client 101/udp          catnet.client.Client
}
# servicecopy
$nb_ build [! h8] {
        udp                     drcl.inet.transport.UDP
        servicecopy 101/udp      catnet.servicecopy.ServiceCopy
}
# servicecopy
$nb_ build [! h9] {
        udp                     drcl.inet.transport.UDP
        servicecopy 101/udp      catnet.servicecopy.ServiceCopy
}
# servicecopy
$nb_ build [! h10] {
        udp                     drcl.inet.transport.UDP
        servicecopy 101/udp      catnet.servicecopy.ServiceCopy
}

! h7/client setHP 7 101
! h4/client setHP 4 101

! h2/resource setHP 2 101
! h5/resource setHP 5 101
! h6/resource setHP 6 101
! h0/resource setHP 0 101

! h3/servicecopy setHP 3 101
! h8/servicecopy setHP 8 101
! h9/servicecopy setHP 9 101
! h10/servicecopy setHP 10 101


puts "Setup static routes..Clients ServiceCopies."

java::call drcl.inet.InetUtil setupRoutes [! h7] [! h3] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h7] [! h8] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h7] [! h9] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h7] [! h10] "bidirection"

java::call drcl.inet.InetUtil setupRoutes [! h4] [! h3] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h4] [! h8] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h4] [! h9] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h4] [! h10] "bidirection"

#puts "routes Client Resources"
java::call drcl.inet.InetUtil setupRoutes [! h7] [! h0] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h7] [! h2] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h7] [! h5] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h7] [! h6] "bidirection"

java::call drcl.inet.InetUtil setupRoutes [! h4] [! h0] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h4] [! h2] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h4] [! h5] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h4] [! h6] "bidirection"

#puts "routes ServiceCopy Resources"
java::call drcl.inet.InetUtil setupRoutes [! h3] [! h0] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h3] [! h2] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h3] [! h5] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h3] [! h6] "bidirection"

java::call drcl.inet.InetUtil setupRoutes [! h8] [! h0] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h8] [! h2] "bidirection"
```

```
java::call drcl.inet.InetUtil setupRoutes [! h8] [! h5] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h8] [! h6] "bidirection"

java::call drcl.inet.InetUtil setupRoutes [! h9] [! h0] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h9] [! h2] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h9] [! h5] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h9] [! h6] "bidirection"

java::call drcl.inet.InetUtil setupRoutes [! h10] [! h0] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h10] [! h2] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h10] [! h5] "bidirection"
java::call drcl.inet.InetUtil setupRoutes [! h10] [! h6] "bidirection"

puts "initialize Client "
! h7/client setKnownResources 4 [java::new long\[\] 4 {0 2 5 6}]
! h4/client setKnownResources 4 [java::new long\[\] 4 {0 2 5 6}]

#Be careful: serviceID 0 stands for bandwidth
puts "initialize ServiceCopies "
! h3/servicecopy setServiceID 3
! h8/servicecopy setServiceID 9
! h9/servicecopy setServiceID 4
! h10/servicecopy setServiceID 2

! h3/servicecopy setPrice 3 23 25
! h8/servicecopy setPrice 1 28 32
! h8/servicecopy setPrice 0 30 50
! h8/servicecopy setPrice 2 26 35

! h9/servicecopy setPrice 4 29 33
! h10/servicecopy setPrice 0 30 35
! h10/servicecopy setPrice 2 300 350

! h3/servicecopy setMoney_balance 1000
! h8/servicecopy setMoney_balance 1000
! h8/servicecopy setMoney_balance 1000
! h8/servicecopy setMoney_balance 1000

puts "initialize Clients "
! h7/client setPrice 2 23 29
! h4/client setPrice 2 270 329
! h7/client setMoney_balance 1000
! h4/client setMoney_balance 1000

puts "initialize Resources"
! h2/resource setPrice 0 12 14
! h5/resource setPrice 0 15 16
! h6/resource setPrice 0 16 38
! h0/resource setPrice 0 45 60
! h2/resource setMoney_balance 1000
! h5/resource setMoney_balance 1000
! h6/resource setMoney_balance 1000
! h0/resource setMoney_balance 1000
#! h0/resource setMyGenotype 40

#Structure n, serviceCopy_ID, service_ID
! h2/resource setKnownServiceCopies 1 [java::new long\[\] 1 {3}] [java::new long\[\] 1 {3}]
! h0/resource setKnownServiceCopies 1 [java::new long\[\] 1 {8}] [java::new long\[\] 1 {3}]
! h5/resource setKnownServiceCopies 1 [java::new long\[\] 1 {9}] [java::new long\[\] 1 {4}]
! h6/resource setKnownServiceCopies 1 [java::new long\[\] 1 {10}] [java::new long\[\] 1 {2}]

#bottleneck bw (from ex_echoer.html)
! n1 setBandwidth 1 1.0e4; # 10Kbps at interface 1
! n1 setBufferSize 1 6000; # ~10 packets at interface 1

puts "Start simulation..."
set time_ 0.001
set sim [attach_simulator .]

puts "Start demand"
# FORMAT: service_Demand(int serviceID, int amount, int transactionId, int duration, int money)
#! h7/client service_Demand 2 3 1 10 50
! h4/client service_Demand 2 3 2 10 50

rt . stop
run .

rt . resume

puts "Stop logging..."
java::call catnet.log.store closelogsocket
```

Annex 2

6.1.1.1 Agent_Source class

catnet.agent

Class Agent_Source

```
java.lang.Object
  |
  +--drcl.DrclObj
        |
        +--drcl.comp.Component
              |
              +--drcl.net.Module
                    |
                    +--drcl.inet.application.SUDPApplication
                          |
                          +--catnet.agent.Agent_Source
```

All Implemented Interfaces:

java.lang.Cloneable, drcl.ObjectDuplicable, java.io.Serializable

Direct Known Subclasses:

Client, Resource, ServiceCopy

Method Summary	
protected abstract boolean	**checkRestrictions**() Description of the Method
void	**dataArriveAtDownPort**(java.lang.Object data_, drcl.comp.Port downPort_) Description of the Method
void	**doAccept_Client**(Msg msg) Description of the Method
void	**doAccept_ServiceCopy**(Msg msg) Description of the Method
int	**doAllocateBW**(Msg msg) Description of the Method
int	**doAllocateStorage**(Msg msg) Description of the Method
void	**doAnswerBWAllocation_LocalResource**(Msg msg) Description of the Method
void	**doAnswerC_Client**(Msg msg) Description of the Method
void	**doApproveProvideService_ServiceCopy**(Msg msg) Description of the Method
void	**doBWAllocation_LocalServiceCopy**(Msg msg) Description of the Method
void	**doCfp_ServiceCopy**(Msg msg) Description of the Method
void	**doCfpAnswer_Resource**(Msg msg) Description of the Method
void	**doCfpC_Client**(Msg msg)

	Description of the Method
void	doCfpCAnswer_ServiceCopy(Msg msg) Description of the Method
void	doConfirm_Resource(Msg msg) Description of the Method
void	doConfirm_Servicecopy(Msg msg) Description of the Method
void	doInform_Resource(Msg msg) Description of the Method
void	doPayment_Client(Msg msg) Description of the Method
void	doPayment_ServiceCopy(Msg msg) Description of the Method
void	doProvideService_Resource(Msg msg) Description of the Method
void	doRequestResource_Client(Msg msg) Description of the Method
int	getPrice() Gets the price attribute of the Agent_Source object
void	goMsg_Dest(int msgType, Msg msg, Hp hp) Description of the Method
protected boolean	interpretAccept(Msg currentMessage) Description of the Method
protected boolean	interpretCfp(Msg currentMessage) Description of the Method
protected boolean	interpretProposal(Msg currentMessage) Description of the Method
protected boolean	interpretReject(Msg currentMessage) Description of the Method
protected abstract void	postAcceptanceMethod() to be implemented in derived classes. defines what to do with the acceptance.
protected abstract void	postRejectanceMethod() Description of the Method
void	setHP(long addr, int port) Sets the hP attribute of the Agent_Source object
void	setPrice(int price) Sets the price attribute of the Agent_Source object
protected void	timeout(java.lang.Object evt_) Description of the Met

Client class

catnet.client

Class Client

```
java.lang.Object
  |
  +--drcl.DrclObj
        |
        +--drcl.comp.Component
              |
              +--drcl.net.Module
                    |
                    +--drcl.inet.application.SUDPApplication
                          |
                          +--catnet.agent.Agent_Source
                                |
                                +--catnet.client.Client
```

All Implemented Interfaces:

java.lang.Cloneable, drcl.ObjectDuplicable, java.io.Serializable

Method Summary	
protected void	`postAcceptanceMethod()` to be implemented in derived classes. defines what to do with the acceptance.
protected void	`postRejectanceMethod()` Description of the Method
protected long	`select_cheapest_servicecopy(java.util.Map r)` Description of the Method
void	`service_Demand(int serviceID, int amount, java.lang.Integer transactionId, int timespan, int money)` Description of the Method
void	`setKnownServiceCopies(int n, long[] res)` Sets the knownServiceCopies attribute of the Client object

Service_Copy class

catnet.servicecopy

Class ServiceCopy

```
java.lang.Object
  |
  +--drcl.DrclObj
        |
        +--drcl.comp.Component
              |
              +--drcl.net.Module
                    |
                    +--drcl.inet.application.SUDPApplication
                          |
                          +--catnet.agent.Agent_Source
                                |
                                +--catnet.servicecopy.ServiceCopy
```

All Implemented Interfaces:

java.lang.Cloneable, drcl.ObjectDuplicable, java.io.Serializable

Method Summary	
protected void	`deallocate_SCBWResource(int transactionid_)` Description of the Method
protected void	`deallocationSCBWTimer(int transactionid_, double timeout_)` Description of the Method
void	`doAccept_Client(Msg msg)` Description of the Method
int	`doAllocateBW(Msg msg)` Description of the Method
void	`doAnswerBWAllocation_LocalResource(Msg msg)` Description of the Method
void	`doAnswerC_Client(Msg msg)` Description of the Method
void	`doCfpAnswer_Resource(Msg msg)` Description of the Method
void	`doCfpC_Client(Msg msg)` Description of the Method
void	`doConfirm_Resource(Msg msg)` Description of the Method
void	`doPayment_Client(Msg msg)` Description of the Method
int	`getPrice()` Gets the price attribute of the Agent_Source object
protected void	`postAcceptanceMethod()` to be implemented in derived classes. defines what to do with the acceptance.
protected void	`postRejectanceMethod()` Description of the Method
void	`requestResource(long resourceAddress, Msg msg)`
protected long	`select_cheapest_resource(java.util.Map r)` Description of the Method
void	`setKnownResources(int n, long[] res)` Sets the knownResources attribute of the ServiceCopy object
void	`setPrice(int price)` Sets the price attribute of the Agent_Source object

Resource class

catnet.resource

Class Resource

```
java.lang.Object
  |
  +--drcl.DrclObj
        |
        +--drcl.comp.Component
              |
              +--drcl.net.Module
                    |
                    +--drcl.inet.application.SUDPApplication
                          |
                          +--catnet.agent.Agent_Source
                                |
                                +--catnet.resource.Resource
```

All Implemented Interfaces:

java.lang.Cloneable, drcl.ObjectDuplicable, java.io.Serializable

Method Summary	
void	`doAccept_ServiceCopy(Msg msg)` Description of the Method
int	`doAllocateStorage(Msg msg)` Description of the Method
void	`doApproveProvideService_ServiceCopy(Msg msg)` Description of the Method
void	`doBWAllocation_LocalServiceCopy(Msg msg)` Description of the Method
void	`doCfp_ServiceCopy(Msg msg)` Description of the Method
void	`doPayment_ServiceCopy(Msg msg)` Description of the Method
int	`getPrice()` Gets the price attribute of the Agent_Source object
protected void	`postAcceptanceMethod()` to be implemented in derived classes. defines what to do with the acceptance.
protected void	`postRejectanceMethod()` Description of the Method
void	`setInitialServiceCopiesatResource(int n, long[] res)` Sets the initialServiceCopiesatResource attribute of the Resource object
void	`setPrice(int price)` Sets the price attribute of the Agent_Source object

Strategy package

catnet.strategy

Class avalanche

```
java.lang.Object
  |
  +--catnet.strategy.avalanche
```

All Implemented Interfaces:

IPricing

Method Summary	
double	**bargain**(java.lang.Integer factorID, int proposalTypeToGenerate, double opponentsCurrentPrice, double myOldPrice, java.lang.Integer negotiationID) Main method of the strategy.
protected boolean	**crossedOffers**(int type, double myPrice, double opPrice) Description of the Method
double	**getAveragePrice**(java.lang.Integer factorID) Gets the averagePrice attribute of the avalanche object
java.lang.String	**getExecutionState**() Gets the executionState attribute of the avalanche object
double	**getInitialPrice**(java.lang.Integer factorID, int type) Gets the initialPrice attribute of the IPricing object, that means the price UP from which a good is BID or DOWN from which a good is ASKed.
double	**getLimitPrice**(java.lang.Integer factorID, int type) Gets the limitPrice attribute of the IPricing object, which means the price UP to which a good is ASKed or DOWN to which a good is BID for selling.
Genotype	**getMyGenotype**() Gets the myGenotype attribute of the avalanche object
NegotiationHistory	**getNegotiationHistory**(java.lang.String negotiationID) Gets the negotiationHistory attribute of the avalanche object
int	**getNegotiationPartner**(java.util.ArrayList groupmembers) Gets the negotiationPartner attribute of the zeroIntelligence object
int	**getRecommendedAction**() Gets the recommendedAction attribute of the avalanche object
void	**getRegressionLines**(java.util.HashMap h) Gets the regressionLines attribute of the zeroIntelligence object
PriceDistribution	**modifyDealRange**(java.lang.Integer factorID, PriceDistribution currentPriceDistribution, int proposalTypeToGenerate) Description of the Method
void	**setLastAgreementPrice**(java.lang.Integer factorID, double price) Sets the lastAgreementPrice attribute of the avalanche object
void	**setMyGenotype**(Genotype myGenotype) Sets the myGenotype attribute of the avalanche object
void	**setParameters**(java.util.HashMap strategy_blueprint) Sets the parameters attribute of the avalanche object
void	**setPrices**(java.lang.Integer factorID, double min, double max) Sets the prices attribute of the avalanche object