Lehrstuhl für Wirtschaftsinformatik
Information Systems Management

No. 10
2006

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik

Gaetano Calabrese (ITC-irst Trento), Björn Schnizler (University of Karlsruhe), Werner Streitberger, Torsten Eymann (University of Bayreuth), Floriano Zini (ITC-irst Trento)

Simulator Development - Annual Report Year 2

Bayreuth Reports on Information Systems Management

UNIVERSITÄT BAYREUTH

ISSN 1864-9300

Authors:

Torsten Eymann (University of Bayreuth)
Werner Streitberger (University of Bayreuth)
Floriano Zini (ITC-irst Trento)
Gaetano Calabrese (ITC-irst Trento)
Björn Schnizler (University of Karlsruhe)

Information Systems and Management Working Paper Series

Edited by:

Prof. Dr. Torsten Eymann

Managing Assistant and Contact:

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

Email: raimund.matros@uni-bayreuth.de

ISSN 1864-9300

IST-FP6-003769 CATNETS
D2.2
Annual Report of WP2

Contractual Date of Delivery to the CEC: 31.08.2006

Actual Date of Delivery to the CEC: 02.10.2006

Author(s): Gaetano Calabrese (ITC-irst Trento), Björn Schnizler (Universität Karlsruhe), Werner Streitberger (Universität Bayreuth), Floriano Zini (ITC-irst Trento)

Workpackage: WP2-Simulation

Est. person months: YY

Security: public

Nature: final

Version: 1.0

Total number of pages: 58

Abstract:

This document describes the activities performed in WP2 - Simulation in the second year of the CATNETS project. In particular, it focuses on the two tasks which WP2 was supposed to work on namely, “Simulator enhancement to support new architecture properties” and “Simulation of application layer networks and refinement”.

Keywords (optional):

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 Marche 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 (BWL VII)
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 Marche Ancona
Dipartimento di Economia
Piazzale Martelli 8
60121 Ancona
Italy
Tel: 39-071- 220.7088 , Fax: +39-071- 220.7102
Contactperson: Mauro Gallegati
E-mail: gallegati@dea.unian.it

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

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

Changes

Version	Date	Author	Changes
0.1	06.06.06	Floriano Zini	Draft structure
0.2	27.09.06	Floriano Zini, Gaetano Calabrese, Björn Schnizler, Werner Streitberger	Almost all material added
0.3	29.09.06	Floriano Zini	Chapters 1, 2, 3, 5 revised
1.0	02.10.06	Floriano Zini, Björn Schnizler	Conclusions + Final revision

Contents

Chapter 1

Introduction

This deliverable describes the activities performed in WP2 - Simulation in the second year of the CATNETS project. According to the workplan of the project, WP2 was supposed to work on two tasks:

Simulator enhancement to support new architecture properties. This tasks involves the enhancement of the Grid simulator OptorSim [opt, BCC$^+$03, CCSM$^+$04], chosen as the base simulation framework for CATNETS (see Deliverable D2.1 [WP205]), in order to permit the simulation of Application Layer Networks (ALNs). The task was supposed to be concluded by the end of year 2.

Simulation of application layer networks and refinement. This tasks relates to the execution of preliminary experiments using the CATNETS simulator in order to do first validation and testing of its implementation. The task will continue in year 3 of the project in order to produce the final version of the CATNETS simulator and substantial evaluation results of the economic service/resource allocation mechanisms under investigation.

The rest of this chapter presents the overall architecture of the simulation process. The following chapters discuss various aspects of the development of the CATNETS simulator. Chapter 2 introduces the ALN model adopted in simulations and the types of service and resource allocation which are simulated. Chapter 3 is about the implementation work: a tool for the generation of ALN scenarios is first described, followed by the description of the enhancements introduced in OptorSim. Chapter 4 describes how a simulation can be set up and executed. Chapter 5 is about relations of WP2 to the other work packages of the project. Finally, Chapter 6 outlines the work to be done in the third year of the project.

1.1 Overall simulation architecture

The CATNETS simulation environment must be able to simulate the behaviour of both the Catallactic distributed service/resource allocation mechanism and the auction-based centralised service/resource allocation mechanism. Both mechanisms are described in deliverables D1.1 [WP105] and D1.2 [WP106].

We adopt a system whose high level workflow is depicted in Figure 1.1. The workflow is executed by has three main components:

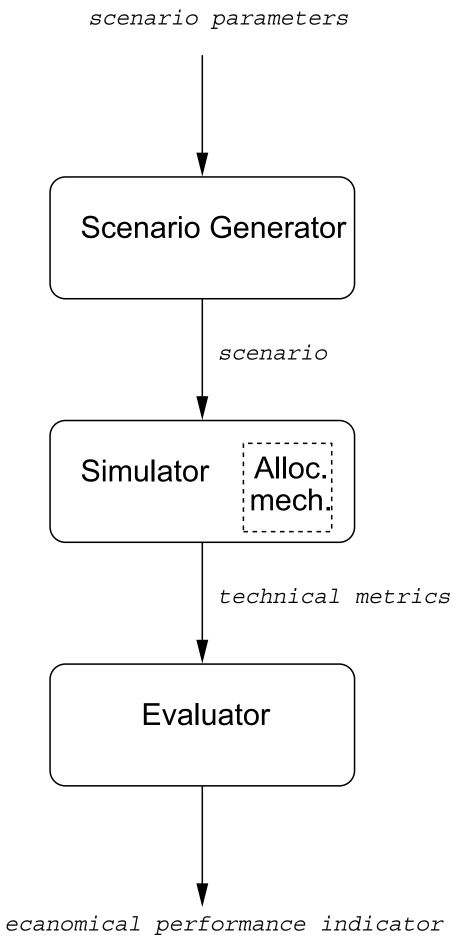


Figure 1.1: Architecture for evaluation system.

Scenario Generator. This component takes a set of scenario parameters as an input and produces a scenario to be simulated as an output.

Simulator. It takes a scenario as an input and execute it by using a pluggable service/resource allocation mechanisms. The output of a simulation is a set of technical metrics as described in Deliverable D4.1 [WP405].

Evaluator. This component takes a set of technical metrics as an input and, as described in Deliverable D4.1 [WP405], calculates an economical performance indicator for the allocation mechanism under observation.

This deliverable focuses on the description of the first 2 components. Details about how the evaluation module is implemented are included in Deliverable D4.2 [WP406].

Chapter 2

Preliminaries

This chapter includes description of:

- the model for ALN to be simulated;
- types of service and resource allocation which are simulated.

2.1 ALN model

An ALN is defined by a connected non-oriented graph

$$ALN = \langle S, L \rangle$$

where $S = \{1, \ldots, n\}$ is a set of sites and $L = \{\langle i_1, j_1 \rangle, \ldots, \langle i_m, j_m \rangle\}$ is a set of network links which connect sites. Each site i is characterised by:

- a *failure probability* fp_i which models the unreliability of the site;
- a triple

  $$\langle CSA_i, BSA_i, RA_i \rangle$$

  where CSA_i is a set of *Complex Service Agents (CSAs)*, BSA_i is a set of *Basic Service Agents (BSAs)*, and RA_i is a set of *Resource Agents (RAs)*. In every site there can be zero or more complex/basic service agents and zero or more resource agents, that is

  $$|CSA_i| \geq 0, |BSA_i| \geq 0, |RA_i| \geq 0$$

  A node with no associated agents is a *router*.

Each link $\langle i, j \rangle$ is characterised by a bandwidth $bw_{i,j}$ which defines the maximum amount of information (bits/second) that can be transmitted along the link.

Complex Service Agents. CSAs are entry points to the ALN and are able to execute *Complex Services (CSs)* for ALN clients. A CS is defined as a set of *Basic Services (BSs)*. CSAs are not specialised: they accept any type of complex service request and take care of the execution of the component basic services.

Basic Service Agents. BSAs provide CSAs with the BSs they need to furnish their complex services to ALN clients. BSs have two *attributes*: *name* and *quality*. Name is a unique identifier whose intended semantics (i.e., the provided service) is shared among all agents. Values for quality are from a discrete set. The intended semantics of quality values is shared among all agents.

For example, there can be a basic service named *pdf_converter* whose quality assumes values in the set $\{bronze, silver, gold, platinum\}$. Another example is a basic service named *PrinterService* with quality in the set $\{silver, gold\}$. Given these BSs, an example CS is

$$CS_1 = \{\langle PDFConverter, gold\rangle, \langle PrinterService, silver\rangle\}$$

As a CS is defined as a set of BS, there there are no assumptions on the order BSs are executed. In other words, the notion of *workflow* is not considered in the definition of CSs.

Every BSA has an associated BS. This means that BSAs are specialised and able to execute specific basic services. Multiple BSAs for the same basic service can co-exist in the ALN. For example, in an ALN there might be two or more BSAs providing a the BS $\langle pdf\_converter, silver\rangle$.

Resources. Resources have a *name*, for example $storage$ or cpu, or ram. Name is a unique identifier whose intended semantics is shared among all agents. Every resource is also characterised by a *quantity* whose value is a positive integer. The intended semantics of quality values is shared among all agents. For example, the resource $storage$ might have $quantity = 50$ while quantities for resources cpu and ram could be 100 and 150 respectively. The unit for each resource is assumed to be virtual: for each resource the value of $quantity$ represents the maximum available amount of resource expressed in the virtual unit.

Resource Bundle. A resource bundle is a set of pairs

$$\langle resource\_name, resource\_quantity\rangle$$

Examples of resource bundles are:

$$RB_1 = \{\langle cpu, 70\rangle, \langle storage, 40\rangle\}$$
$$RB_2 = \{\langle cpu, 50\rangle, \langle storage, 50\rangle, \langle ram, 75\rangle\}$$

Every basic service has an associated resource bundle. The bundle defines which resources and respective quantity are necessary for service provision. For example, the association

$$\langle PDFConverter, gold\rangle \longrightarrow RB_1$$

specifies that for the execution of a pdf conversion having quality gold there is the need of a 70 CPU units and 40 storage units.

Resource Agents. RAs are "proxies" for aggregations of resources. Their task is to provide BSAs with resources needed for the execution of basic services. Every RA has an associated *available resource bundle*. An available resource bundle is defined similarly to a resource bundle but the resource quantities defines the maximum resource amounts which are available from the RA. Example of available resource bundles are

$$ARB_1 = \{\langle cpu, 100\rangle, \langle storage, 40\rangle\}$$
$$ARB_2 = \{\langle cpu, 100\rangle, \langle storage, 40\rangle, \langle ram, 150\rangle\}$$

Multiple RAs for the same available resource bundle can co-exist in the ALN.

2.2 Service and resource allocation

The allocation of services and resource takes place in two separate markets, the *service market* and the *resource market*. As the meaning of "allocation" in the two markets is not the same, this section describes its two semantics.

2.2.1 Service Market

Basic services are provided by single BSAs. A CSA receiving a request for CS provision starts a "service allocation process" for each of the BSs included in the CS. Every "service allocation process" produces the selection of a single BSA able to provide the BS. Service allocation abortion is possible.

2.2.2 Resource Market

A BSA requests a bundle of resources on the resource market in order to be able to execute its specific basic service. Issuing such a request initiates a "resource bundle allocation process". This process can have a number of outcomes:

1. abortion, if there are no RAs able to provide the needed resources;
2. one single RA provides the total amount of every resource in the requested bundle;
3. multiple RAs partially provide all resources in the requested bundle;

Example. Assume there is a BSA providing the basic service

$$BS_1 = \langle PDFConverter, gold \rangle$$

and the resources bundle

$$RB_1 = \{\langle cpu, 70\rangle, \langle storage, 40\rangle\}$$

associated to BS_1 as follows

$$BS_1 \longrightarrow RB_1$$

In addition, assume there are two resource providers RA_1 and RA_2 associated with the following available resource bundles and two RA-ARB associations

$$\begin{array}{lcl} RA_1 & \longrightarrow & ARB_1 = \{\langle cpu, 100\rangle, \langle storage, 40\rangle\} \\ RA_2 & \longrightarrow & ARB_2 = \{\langle cpu, 100\rangle, \langle storage, 40\rangle, \langle ram, 150\rangle\} \end{array}$$

A resource bundle allocation process started by BSA_1 might result into:

1. abortion
2. the single provider allocation

   $$RA_1(\{\langle cpu, 70\rangle, \langle storage, 40\rangle\}) \longrightarrow BSA_1$$

   where the notation $RA_1(\{\langle cpu, 70\rangle, \langle storage, 40\rangle\})$ means that RA_1 provides 70 cpu units and 40 storage units;
3. the multi-provider allocation

   $$RA_1(\{\langle cpu, 30\rangle, \langle storage, 20\rangle\}), RA_2(\{\langle cpu, 40\rangle, \langle storage, 20\rangle\}) \longrightarrow BSA_1$$

4. the multi-provider allocation

   $$RA_1(\{\langle cpu, 40\rangle, \langle storage, 20\rangle\}), RA_2(\{\langle cpu, 0\rangle, \langle storage, 20\rangle\}) \longrightarrow BSA_1$$

Chapter 3

Simulator enhancement

This chapter describes the first two components of the simulation/evaluation framework depicted in Figure 1.1. They are the *Scenario Generator* end the *Simulator*. The description of the third component, namely the *Evaluator* is included in Deliverable D4.2 [WP406].

3.1 Scenario generator

The generation of a scenario can be done in two ways:

- manually, using a GUI which allows the user to specify the topology of the ALN and the set of agents located on every ALN site.
- using an automated procedure which takes a set of ALN requirements as input and generates a topology, as well as a distribution of the agents on sites, satisfying the given requirements.

3.1.1 Manual generation: requirements

The manual generation can be used whenever the number of ALN sites is in the order of few tens and the number of agents in every site is at most few units. The manual generator provides the user with a GUI and a graphical tool for the input of the ALN specification and the generation of the files containing the ALN description according to the OptorSim format.

The manual generator helps the user in the following activities:

- definition of Resources, Basic Services, and Complex Services;

- creation of an ALN topology;
- association of CSAs, BSAs, RAs and properties to ALN nodes;
- association of bandwidth values to ALN links;
- generation of the output files in OptorSim format.

Definition of Resources. A Resource is univoquely defined by its name. Within the tool it will be possible to define Resource names, to check which names have been defined and possibly to delete them.

Definition of Basic Services. A Basic Service is defined by a name and a quality (chosen among a defined set). Each Basic Service is associated to a Resource Bundle and this is defined by a set of pairs $\langle Resource, Quantity \rangle$. Tool has to give the possibility of inserting the Basic Service name, to choose the quality from a predefined set, to define the Resource Bundle choosing the Resources from a list and set the quantity for each of them. When the Basic Service is created, a unique identifier and the Resource Bundle will be assigned to it. The created Basic Services have to be shown in a list, so it will be possible to check them and eventually to delete them.

Definition of Complex Services. A Complex Service is defined by its name, by a set of Basic Service and by its execution probability. The tool has to give the possibility to insert the Complex Service name and to visualize a list of Basic Service. From this list it will be possible to choose the Basic Services that compose the Complex Service. A unique identifier will be created and assigned to the Complex Service. It will be possible to visualize the list of created Complex Services, to check them and to delete them. There will be also a field where to set the execution probability.

ALN creation. The manual scenario generator has to provide a graphical editor able to create the nodes and the edges of an ALN. The requirements for this editor are:

- the source code should be available, so it will be possible to customize the application;
- it should be written in Java to allow an easier integration with OptorSim;
- it should allow the user to save the graphs in a text file format, so that it will be easy to convert them in a format suitable for OptorSim;

After a preliminary search we found some open source tool with these features: JUNG, Colt, JGraphpad, Jgapth and OpenJGraph. So we decided to adopt one of these and integrate it into the main tool.

Association of agent and properties to ALN nodes. Once the ALN topology is created, the tool has to allow the user to associate agents and properties to nodes.

For each node, the tool has to allow to:

- set the failure probability;
- set the number of CSAs;
- set the complex service schedule policy[1];
- set the BSAs and select for each of them the Basic Service they provide;
- set the RAs and related Available Resource Bundles; each ARB is set by selecting the component resources and defining their maximum quantity;
- if a scenario for the simulation of the centralised mechanism is being set, there will be the possibility of deciding if the node contains a agents wrapping the service market central auctioneer or the resource market central auctioneer.

For each edge in the land, the tool should be able to define also the value of its bandwidth.

Generation of output in OptorSim format. Once the definitions of ALN and Agents are finished, the tool has to generate an output in the form of four text file according to the OptorSimformat. For details about the structure of this file, see Section 4.1.

3.1.2 Automatic generation: requirements

The automatic scenario generator should be based on an algorithm which takes a set of ALN parameters as input and generates all the objects which in the manual generator are defined by hand by the user. Moreover, the automatic generator should allow for an automatic distribution of agents over sites by using a parametrised policy. Finally, the automatic generator should be able to generate the scenario configuration in OptorSim format. Figure 3.1 shows the main phases of the automatic generation.

The figure is self-explanatory for the first phase. In the following, the requirements for the other phases are detailed.

[1]The schedule policy define which complex service are potentially executable by CSAs located in the node.

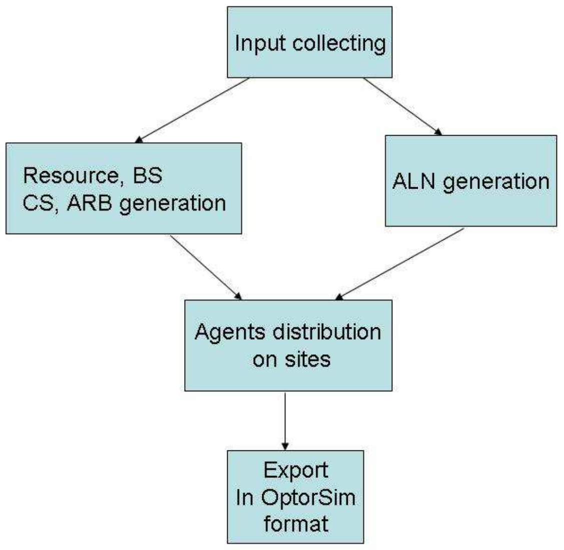


Figure 3.1: Main phases of the automatic topology generator.

ALN generation. We decided to base the generation of an ALN topology on an existing tool for automatic graph generation. The requirements for such a tool are:

- generation of various regular network topologies (i.e ring, mash, hypercube);
- generation of output in text format, to facilitate the generation of files needed as input for OptorSim;
- open source code written in Java.

We evaluated some open source tools with respect these features: BRITE, GT-ITM, and Inet. BRITE, GT-ITM are more complete from the point of view of the type of topologies which can be generated. Moreover, BRITE has a version written in Java. Therefore, BRITE was chosen as the base tool.

Resource generation. The parameters driving the automatic generation of resources are:

- the number n_r of resources to be generated;
- the maximum admissible quantity for each resource.

The quantity of each resource is a random integer number smaller then the maximum. The name of each resource is generated using numbers in ascending order, i.e $r_1 \dots r_{n_r}$.

Available Resource Bundles generation. The parameter driving the automatic generation of ARBs are:

- maximum cardinality (number of resources) m for ARB;
- number n_{arb} of ARBs to be generated.

The number of resources composing each ARB is a random number (generated with a uniform probability) between 0 and the maximum cardinality m.

Basic Service generation. For the generation of basic services, the parameter are:

- number n_{bs} of BSs to be generated;
- quality, randomly chosen with uniform probability from a finite set of names having shared intended meaning.

The names of the Basic Service are generated following an ascending order, i.e $bs_1 \dots bs_{n_{bs}}$. To associate the Basic Service with the Resource Bundle, first we choose the cardinality of the bundle using a random number N, then we choose the N Resources with uniform probability. Then we define the quantity of each resource using a random value less then or equal to the maximum quantity available for that Resource.

Complex Service Generation. For the generation of basic services, the parameter are:

- number n_{cs} of CSs to be generated;
- maximum cardinality m for the CSs.

The names of the Complex Service are generated using numbers in ascending order, i.e $cs_1 \dots cs_{n_{cs}}$

The cardinality of each CS is a random number less then or equal to m. The set of BSs to each CS is defined by randomly choosing a number of BSs equal to the cardinality of the CS. The execution probability for each CS is equal to $1/n_{cs}$.

Agent distribution on sites.

3.1.3 Scenario Generator: implementation

Implementation of the manual generator

The implemented manual generator is described by means of an example. Suppose we want to set up a scenario corresponding to an ALN with 4 sites (S1, S2, S3, S4) as shown in Figure 3.2. Each site contains zero o more CSAs, BSA, or RAs. When requested a, a

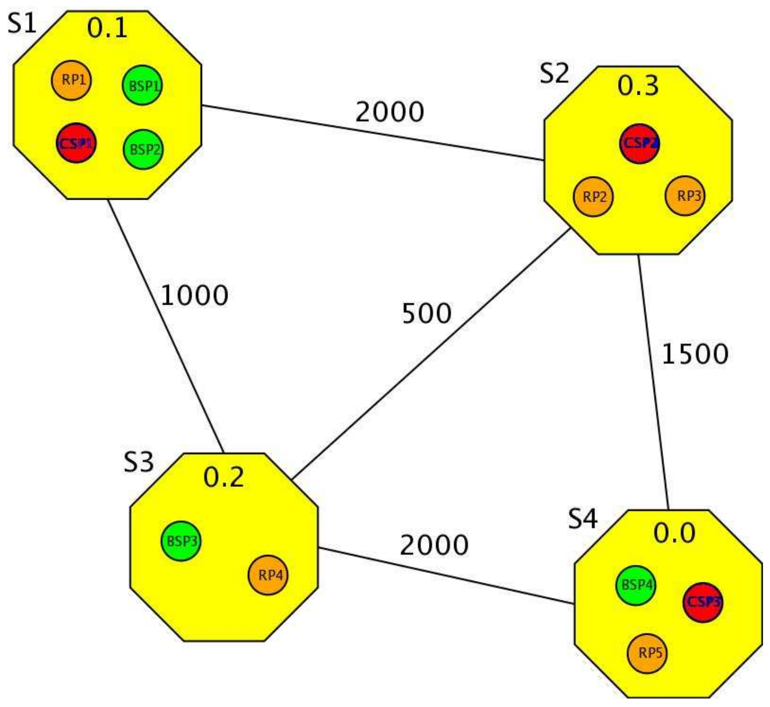


Figure 3.2: An example ALN.

site could not be able to provide any service or resource and this is defined using a failure Probability (FP). For example, the site S1 has one Complex Service Agent, two Basic Service Agents, one Resource Provider Agent and a failure probability of 0.1. The label on each edge represent the bandwidth of the ALN link.

In the following, the functions provided by the GUI are explained.

Definition of Resources. When the scenario generator is started, the GUI shown in Figure 3.3 appears on the screen. Resource names are defined by inserting a string denoting the name in the *Resource* tab and then pressing the *Add* button. The name will be added into the list. To delete a resource name from the list, simply select it and press the *Delete* button.

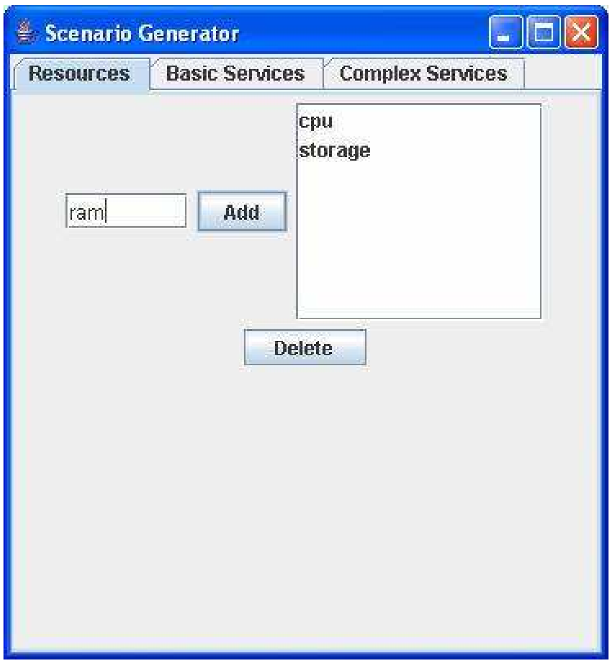


Figure 3.3: Resource definition.

Definition of Basic Services, Figure 3.4 shows the interface to define the Basic Services (BSs). The figure relates to the specification of the BSs described in table 3.1. The

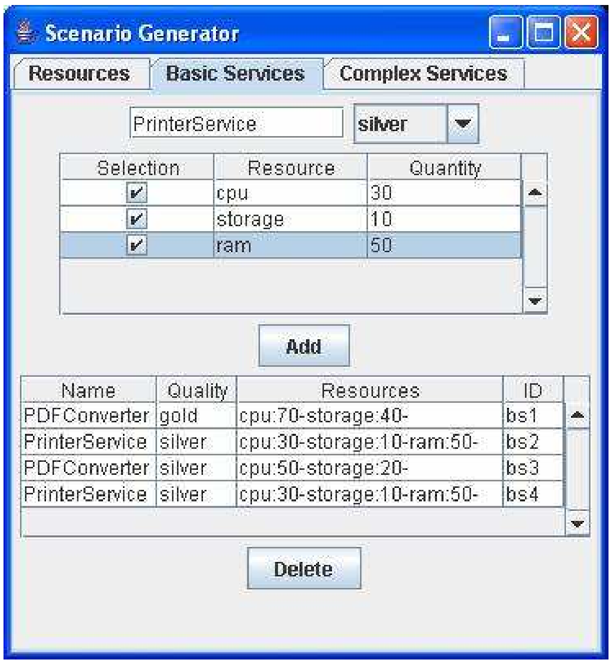
name = PDFConverter	quality = gold	RB = (cpu 70, storage 40)
name = PrinterService	quality = silver	RB = (cpu 30, storage 10, ram 50)
name = PDFConverter	quality = silver	RB = (cpu 50, storage 20)
name = PrinterService	quality = silver	RB = (cpu 30, storage 10, ram 50)

Table 3.1: Example BSs.

specification of a BS (e.g. *PDFConverter*) is done by: (1) inserting the name in the text field, (2) selecting its quality from the combo box (2) choosing the resources that make up the Resource Bundle, (3) assigning the quantity to each resource, and finally (4) pressing the *Add* button.

To delete a BS, simply select it from the table and press the button *Delete*.

Definition of Complex Services. Figure 3.5 shows the GUI for definition of CSs. The figure relates to the specification of the CSs described in table 3.2. The specification of a CS (e.g. cs4) is done by: (1) setting up the CS execution probability *ep*, (2) selecting the Basic Service IDs that make up the Complex Service (for a multiple choice keep pressed the CTRL button), and (3) press the *Add* button.

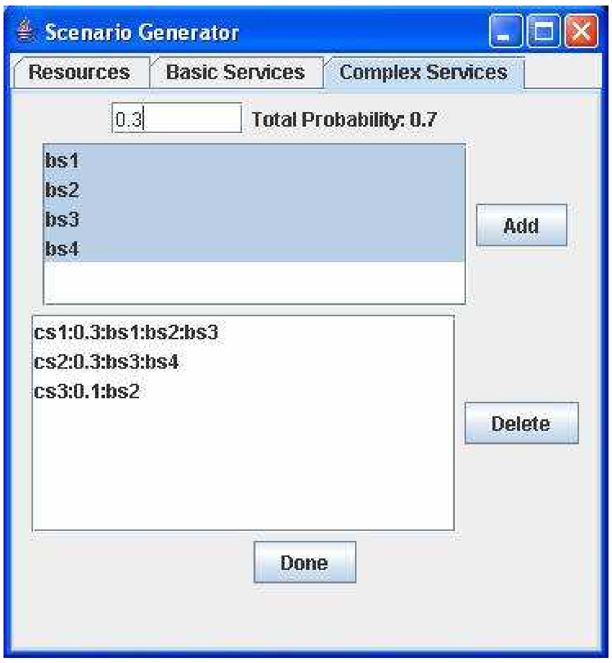


PrinterService silver

Selection	Resource	Quantity
✓	cpu	30
✓	storage	10
✓	ram	50

Name	Quality	Resources	ID
PDFConverter	gold	cpu:70-storage:40-	bs1
PrinterService	silver	cpu:30-storage:10-ram:50-	bs2
PDFConverter	silver	cpu:50-storage:20-	bs3
PrinterService	silver	cpu:30-storage:10-ram:50-	bs4

Figure 3.4: Basic Service definition.



0.3 Total Probability: 0.7

bs1
bs2
bs3
bs4

cs1:0.3:bs1:bs2:bs3
cs2:0.3:bs3:bs4
cs3:0.1:bs2

Figure 3.5: Complex Service definition.

To delete a Complex Service, select it from the list and press the *Delete* button.

name = cs1	BSs = {bs1, bs2, bs3}	ep = 0.3
name = cs2	BSs = {bs3, bs4}	ep = 0.2
name = cs3	BSs = { bs2}	ep = 0.2
name = cs4	BSs = { bs1, bs2, bs3, bs4}	ep = 0.3

Table 3.2: Example CSs.

It is possible to go back and modify the choices made previously. Once all the parameter are correct, press the button *Done*. The GUI will be closed, and will appear the editor for drawing the ALN topology.

Creation of an ALN topology. As ALN editor we have chosen OpenJGraph, an open source tool written in Java. We have modified the code to implement the necessary features.

Figure 3.6 shows how ALN nodes and links are created. For node creation, click on the button with the oval shape in the tool bar and then click on the client area.

To connect two nodes with a link, select the button *simple edge* on the tool bar an then draw the line.

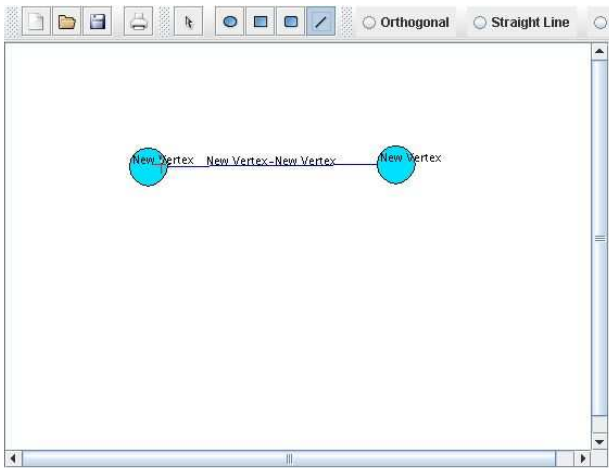


Figure 3.6: Node and link creation.

To modify the name of a node right click on it, select *Vertex Properties* and modify the *tab label*, as shown in Figure 3.7 and Figure 3.8.

To set up the bandwidth on the edges, right click on it, select *Edges Properties*, as shown in Figure 3.9, unselect the check box, and write the bandwidth value. Figure 3.10 shows the created ALN.

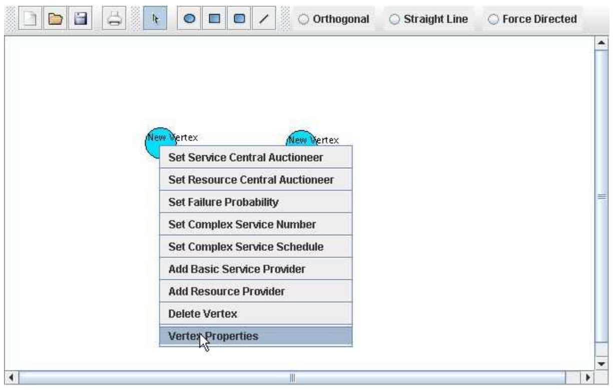
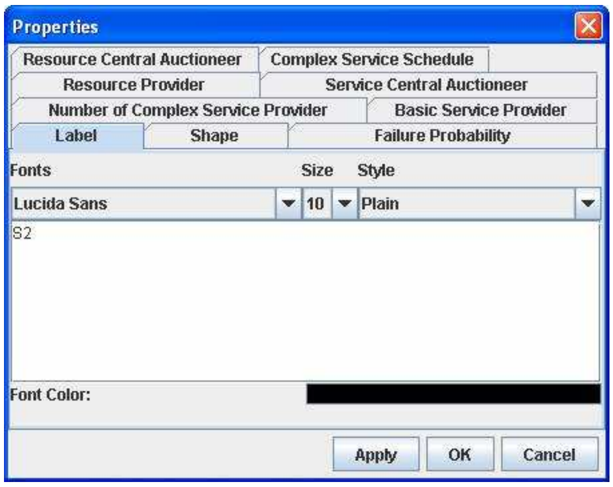

Orthogonal
Straight Line
Force Directed
New Vertex
New Vertex
Set Service Central Auctioneer
Set Resource Central Auctioneer
Set Failure Probability
Set Complex Service Number
Set Complex Service Schedule
Add Basic Service Provider
Add Resource Provider
Delete Vertex
Vertex Properties

Figure 3.7: Set up the Vertex name.



Fonts: Lucida Sans Size: 10 Style: Plain

S2

Font Color:

Figure 3.8: Set up the Vertex name.

Association of Agents and properties to the ALN nodes. Once defined the ALN, we can set up the agents on the sites. Right clicking on a Node, a menu will appear. As shown in the Figure 3.11 the possible choices are the following:

Set Service Market Central auctioneer. In the centralised market, we can decide in which site the agent wrapping the central auctioneer is located.

Set Resource market Central Auctioneer. In the centralized market, we can decide in which site the agent wrapping the central auctioneer is located.

Set Failure Probability. Set the failure probability of the site.

Set number of CSAs. Set the number of Complex Service Agents present on the site.

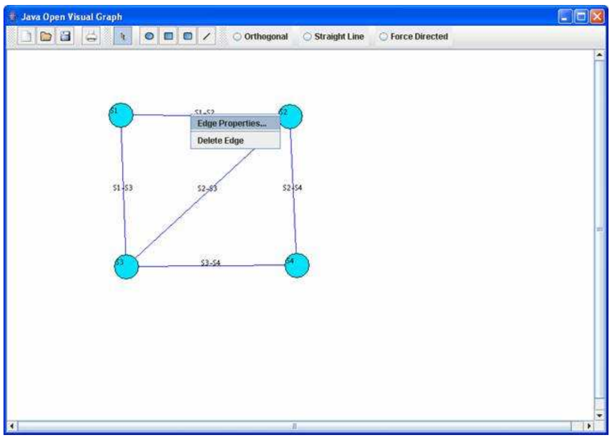


Figure 3.9: Set up the bandwidth.

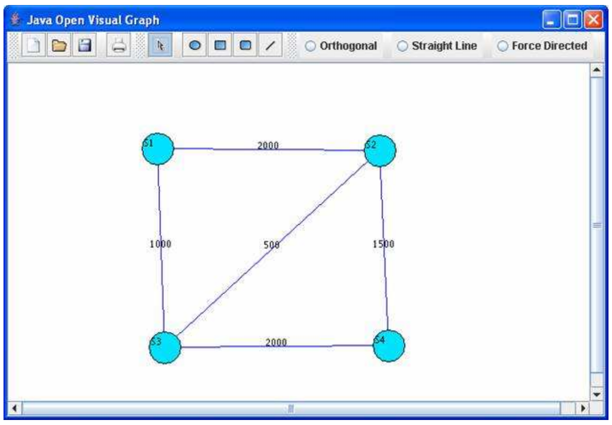


Figure 3.10: The ALN.

Set Complex Service Schedule. Select a set of Complex Service that will run on that site (see Figure 3.12)

Select BSAs. Select the set of Basic Service Agents that will run on the site (see Figure3.13

Select RAs. Add Resource Agents to the site and define the related Available Resource Bundle (see Figure 3.14).

Generation of the output files in OptorSim format. Figure 3.15 shows how to generate abs save the scenario configuration files in OptorSim format. Press the *Save* button from the toolbar, a dialog will appear, choose the directory where to save the files, and press *Save*.

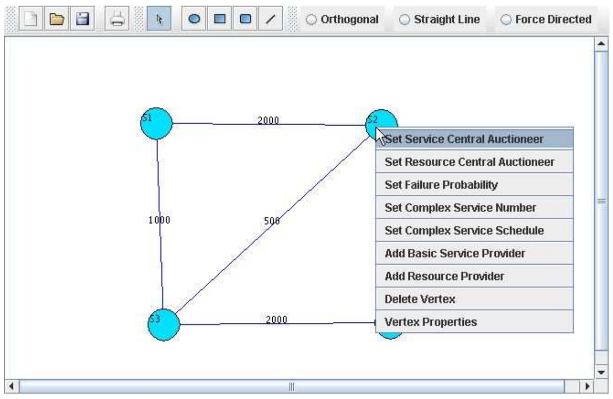
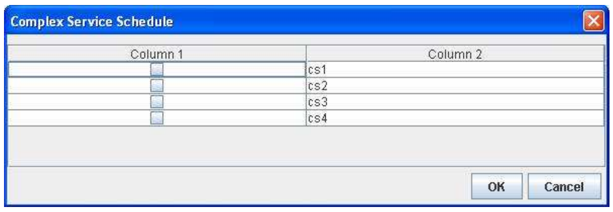
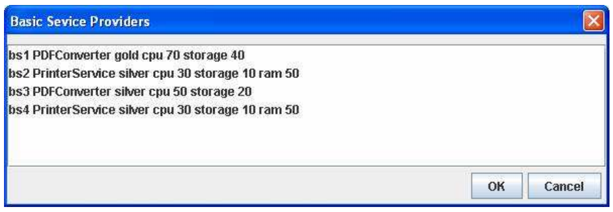

Orthogonal
Straight Line
Force Directed
S1
S2
S3
2000
1000
500
2000
Set Service Central Auctioneer
Set Resource Central Auctioneer
Set Failure Probability
Set Complex Service Number
Set Complex Service Schedule
Add Basic Service Provider
Add Resource Provider
Delete Vertex
Vertex Properties

Figure 3.11: Set up the Agents.



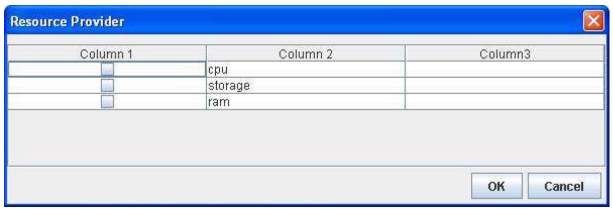
Column 1	Column 2
☐	cs1
☐	cs2
☐	cs3
☐	cs4

Figure 3.12: Set up Complex Service Schedule.



bs1 PDFConverter gold cpu 70 storage 40
bs2 PrinterService silver cpu 30 storage 10 ram 50
bs3 PDFConverter silver cpu 50 storage 20
bs4 PrinterService silver cpu 30 storage 10 ram 50

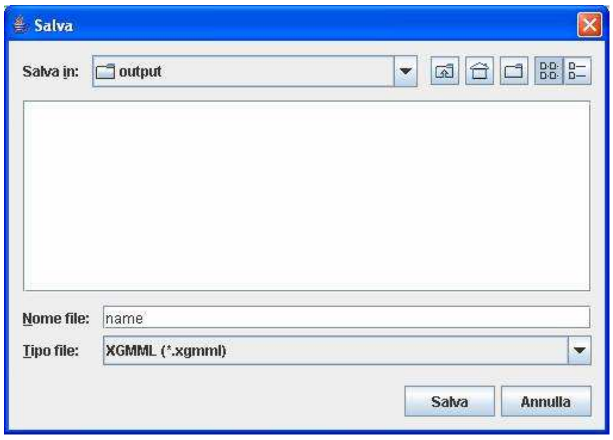
Figure 3.13: Set up Basic Service Agents.



Column 1	Column 2	Column3
☐	cpu	
☐	storage	
☐	ram	

Figure 3.14: Set up Resource Agents.

Four files are generated: `arb.conf`, `bs.conf`, `cs.conf` and `topology.conf`. Examples of these files and explanation of their content is given in Chapter 4.

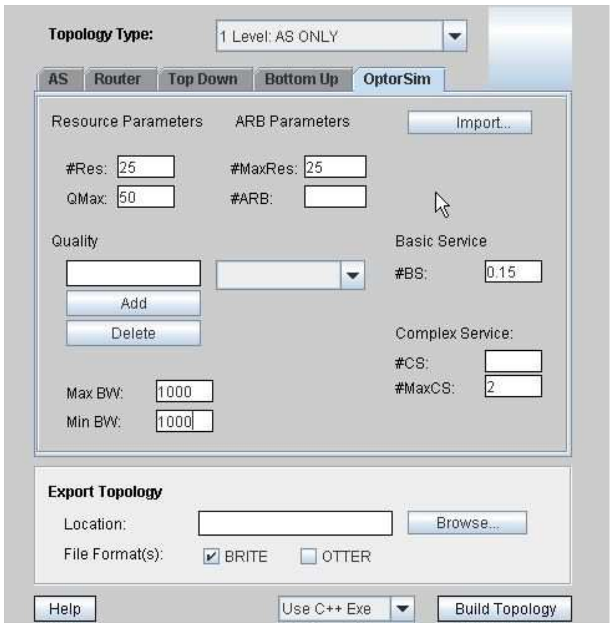


Nome file: name
Tipo file: XGMML (*.xgmml)

Figure 3.15: Save the configuration file.

3.1.4 Automatic Generation: implementation

The scenario Automatic Generator is currently under development. It will be completed in the 3rd year of the CATNETS project. As already mentioned, we decided to use BRITE as the base tool for topology automatic generation. BRITE is composed by a GUI where to define the parameters and by a topology generator. We have already integrated a new interface where it is possible to define all the parameters needed for the automatic generation of ALN scenarios. The modified BRITE GUI is shown in Figure 3.16.



Topology Type: 1 Level: AS ONLY
#Res: 25
QMax: 50
#MaxRes: 25
#ARB:
#BS: 0.15
#CS:
#MaxCS: 2
Max BW: 1000
Min BW: 1000
Location:
File Format(s): [✓ BRITE] OTTER
Use C++ Exe

Figure 3.16: BRITE extended with the OptorSim parameter tab.

An explanation of the parameters which can be set using the GUI follows.

Resource Parameters. They are: **#Res**: number of Resources that have to be generated; **QMax**: Maximum quantity for each resource.

ARB Parameters. They are: **#MaxRes**: the maximum number of resources composing the ARB;**#ARB**: number of ARBs to be generated.

Quality. It is possible to define the values for the quality adding them to a combo box.

Basic Service. **#BS** is number of Basic Services that have to be generated.

Complex Service. The parameters are: **#CS**: number of Complex Services that have to be generated; **#MaxCS**: maximum number of Basic service composing the Complex Service.

Bandwidth. The parameters are: **Max BW**: maximum value for the bandwidth; **Min BW**: minimum value for the bandwidth.

3.2 Simulator

The CATNETS simulator is a tool written in Java, based on OptorSim [opt, BCC$^+$03, CCSM$^+$04], a Data Grid simulator which was initially developed in the framework of the European DataGrid project [edg].

The goal of the CATNETS simulator is to allow experimentation with and evaluation of 2 economy-bases service and resource allocation mechanisms to be used in Application Layer Networks. Given as input: (1) an ALN configuration (produced by the *Scenario Generator* described in Section 3.1), (2) a set of simulation parameters included into a configuration file, and (3) an allocation mechanism (*centralised* or *catallactic*, the CATNETS simulator runs a number of *Complex Services* on the simulated ALN. During simulation it records metrics used for an off-line evaluation of the mechanism performance.

This section gives an overview of the architecture of the simulator and a high level description of the packages in which the code is structured. More details are given on how the integration of the centralised and catallactic mechanisms into OptorSim has been performed.

3.2.1 Simulator architecture

The architecture of the CATNETS simulator is depicted in Figure 3.17. The component at the top of the figure simulates ALN users who submits requests for the execution of Complex Services. In the simulator, this component is implemented as a thread. The

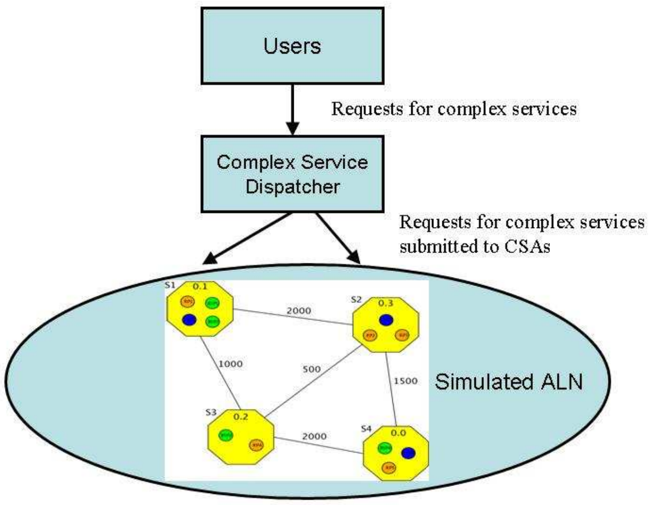

Users
Requests for complex services
Complex Service Dispatcher
Requests for complex services submitted to CSAs
S1
0.1
S2
0.3
S3
0.2
S4
0.0
2000
1000
500
1500
2000
Simulated ALN

Figure 3.17: High level architecture of the CATNETS simulator.

sequence of submitted requests is determined by a specific *pattern*, which is a parameter of the simulation. Several patterns are available, as described in Section 4.1.5.

The component called *ComplexServiceDispatcher* performs the dispatching of complex service requests to Complex Service Agents (CSAs). In the simulator, this component is implemented as a thread. Various dispatching policies are available, as as described in Section 4.1.5.

The bottom of Figure 3.17 represents the simulated ALN. In the simulator, Complex Service Agents, Basic Service Agents (BSAs), and Resource Agents (RAs) are implemented as threads. In every simulated ALN site there is also a components called *P2P mediator*, whose task is to manage the exchanging of messages between ALN sites. P2P Mediators are also implemented as threads.

The structure of the simulated ALN differs depending on the allocation mechanism. When the central mechanisms is adopted, the structure is shown in Figure 3.18. The ALN includes a special site where only the central auctioneers are located. The acronym SMAA stands for *Service Market Auctioneer Agent*, while RMAA for *Resource Market Auctioneer Agent*. This site is fully connected to all other ALN sites and all the allocation requests for services or resources are sent to it via the P2P mediators.

When the catallactic decentralised mechanism is simulated, the special site is not present in the ALN. As shown in Figure 3.19, in this case P2P Mediators perform a propagation of messages over the ALN and agent interaction is likely to a "classical" P2P

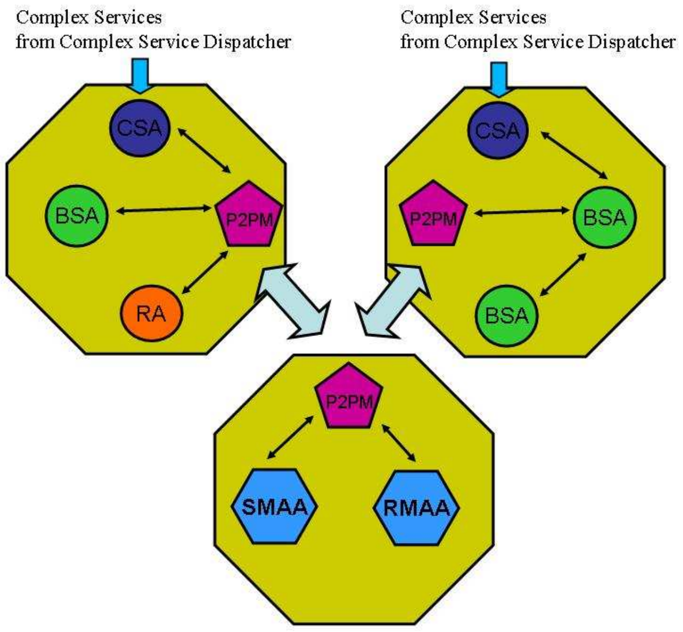


Figure 3.18: ALN structure for centralised mechanism.

model.

3.2.2 Agent behaviour

The behaviour of users and agents in the ALN depends on the adopted service/resource allocation mechanism.

Catallactic mechanism. When this mechanism is adopted the behaviour of ALN users is proactive while CSAs, BSAs, and RAs are reactive. The intended meaning of proactive is that users or agents are able to "take the initiative" in the interaction with other agents while by reactive we mean that they respond to external messages from other agents or users.

The behaviour of users and agents can be summarised as follows:

- ALN users proactively submit requests for complex services to CSAs;
- CSAs react to a requests for complex service by issuing a sequence of requests for the involved basic services over the ALN; they are also able to react to messages from BSAs while bargaining for basic services;

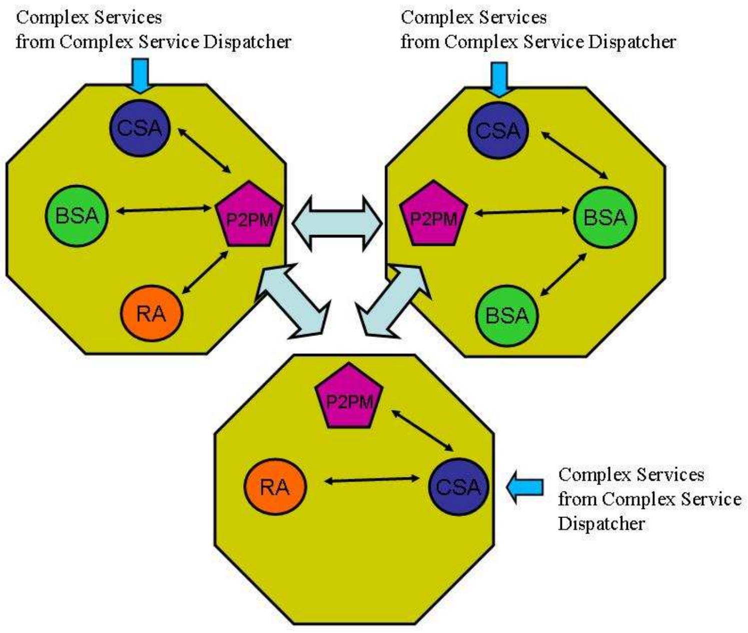


Figure 3.19: ALN structure for catallactic mechanism.

- the behaviour of BSAs is also reactive: they (1) respond to request for basic services by possibly generating an offer for the requested basic service, (2) react to message from CSAs while bargaining for basic services, and (3) issuing requests for the associated resource bundle after a negotiation for a basic service ends successfully.
- the behaviour of RAs is reactive: they (1) respond to request for resource bundles form BSAs by possibly generating an offer for the whole or part of the bundle and (2) react to message from BSAs while bargaining for resources.

Centralised mechanism. When this mechanism is adopted users and all agents in the ALN but CSAs act proactively. In both the service and resource markets the matching of requests and offers is performed by a dedicated centralised auctioneer.

- As when the catallactic mechanism is used, ALN users proactively submit requests for complex services to CSAs;
- CSAs react by submitting requests for the involved basic services to the centralised service auctioneer (SMAA);
- BSAs which want to sell a basic service proactively send offers for this service to the centralised service auctioneer (SMAA). Moreover, when a their service is successfully allocated, they submit requests for resource bundles to the centralised

resource auctioneer (RMAA) in order to get the resources which are necessary for service provision.

- RAs proactively send offers for resource bundles to the resource central auctioneer.

3.2.3 Overview of the code structure

The development of the CATNETS simulator is done using Eclipse SDK [ecl]. Figure 3.20 shows the structure of the simulator code. In the following a brief description of the main packages is given.

Package *edu*. This packages includes the code implementing the MACE algorithm on which the centralised resource allocation mechanism is based. Details on MACE are given in Deliverable D1.1 [WP105]. The package also includes code implementing the continuous double auction adopted for the centralised service allocation mechanism and some statistical distribution.

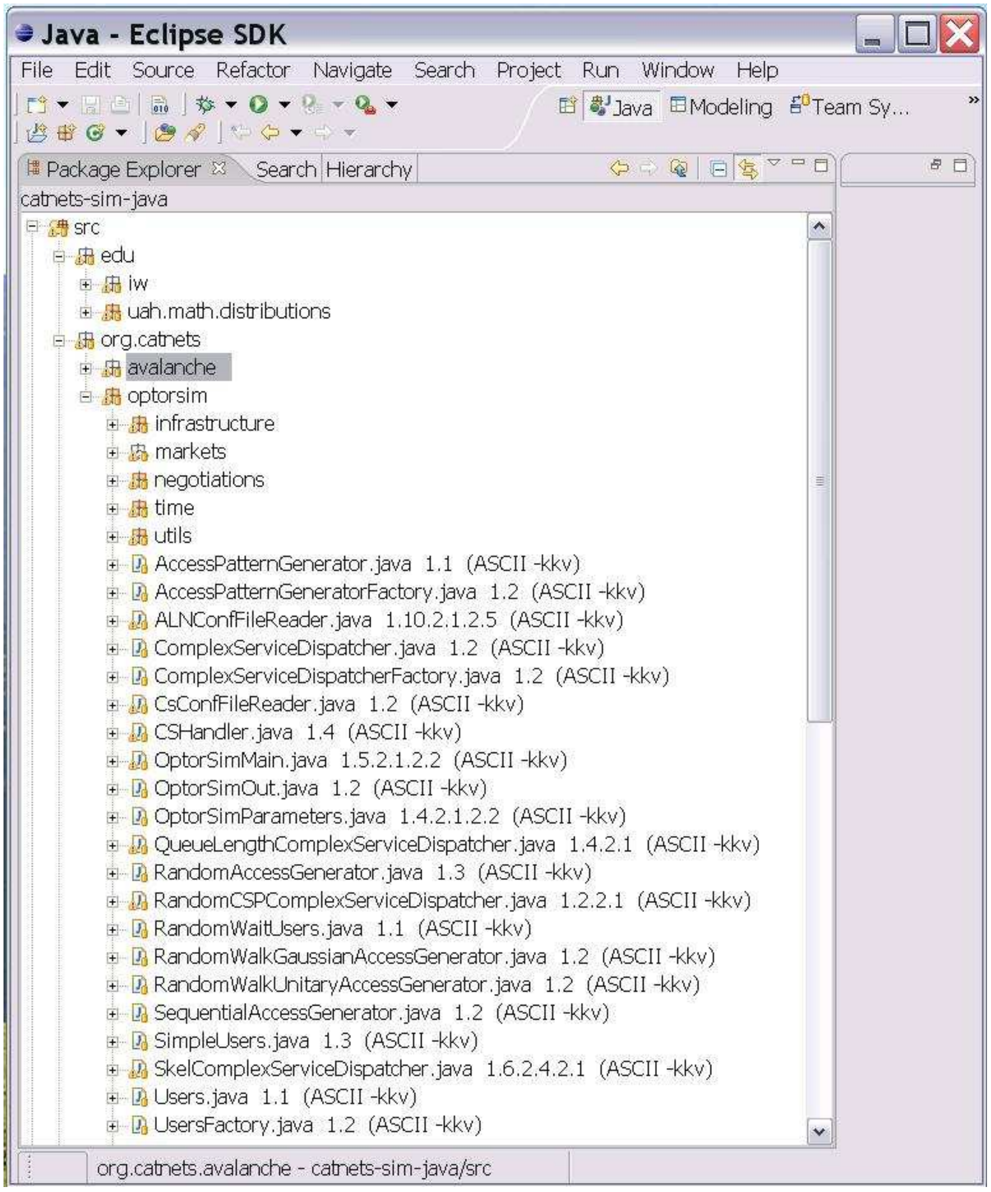
Package *org.catnets.avalanche*. This package includes all the classes implementing the strategy used by agents when the catallactic allocation mechanism is adopted. Details on the strategy are given in Deliverable D1.1 [WP105] and Deliverable D1.2 [WP106].

Package *org.catnets.optorsim*. The classes in this package implement the parsing of the simulation configuration files and the instantiation of the object described in the files. For example, the class *ALNConfFileReader* takes the file describing an ALN topology as an input and instantiate objects for all ALN sites, including components corresponding to agents located in the sites.

This package also contain classes implementing the components *Users* and *ComplexServiceDispatcher* shown in Figure 3.17, as well as a number of policies which drive the behaviour of the two threads. Details are given in Section 4.1.5.

Package *org.catnets.optorsim.infrastructure*. The package includes classes needed for the implementation of an ALN as a graph of sites interconnected by network links. Notable classes are *ALNContainer*, which a run time is a singleton embedding all the ALN sites, and *ALNSite*, whose instances implements sites and their sub-components.

Package *org.catnets.optorsim.markets*. The classes in this package relate to the two simulated market models. As far as the catallactic market is concerned, the package includes three classes implementing the component *CatallacticReasoner*, embedded in agents acting on the market. A catallactic reasoner is a wrapper developed to make the


Java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java Modeling Team Sy...
Package Explorer Search Hierarchy

```
catnets-sim-java
src
  edu
    iw
    uah.math.distributions
  org.catnets
    avalanche
    optorsim
      infrastructure
      markets
      negotiations
      time
      utils
      AccessPatternGenerator.java 1.1 (ASCII -kkv)
      AccessPatternGeneratorFactory.java 1.2 (ASCII -kkv)
      ALNConfFileReader.java 1.10.2.1.2.5 (ASCII -kkv)
      ComplexServiceDispatcher.java 1.2 (ASCII -kkv)
      ComplexServiceDispatcherFactory.java 1.2 (ASCII -kkv)
      CsConfFileReader.java 1.2 (ASCII -kkv)
      CSHandler.java 1.4 (ASCII -kkv)
      OptorSimMain.java 1.5.2.1.2.2 (ASCII -kkv)
      OptorSimOut.java 1.2 (ASCII -kkv)
      OptorSimParameters.java 1.4.2.1.2.2 (ASCII -kkv)
      QueueLengthComplexServiceDispatcher.java 1.4.2.1 (ASCII -kkv)
      RandomAccessGenerator.java 1.3 (ASCII -kkv)
      RandomCSPComplexServiceDispatcher.java 1.2.2.1 (ASCII -kkv)
      RandomWaitUsers.java 1.1 (ASCII -kkv)
      RandomWalkGaussianAccessGenerator.java 1.2 (ASCII -kkv)
      RandomWalkUnitaryAccessGenerator.java 1.2 (ASCII -kkv)
      SequentialAccessGenerator.java 1.2 (ASCII -kkv)
      SimpleUsers.java 1.3 (ASCII -kkv)
      SkelComplexServiceDispatcher.java 1.6.2.4.2.1 (ASCII -kkv)
      Users.java 1.1 (ASCII -kkv)
      UsersFactory.java 1.2 (ASCII -kkv)
org.catnets.avalanche - catnets-sim-java/src
```

Figure 3.20: Simulator package structure.

economic strategy included in *org.catnets.avalanche* available to agents. As for the centralised market, the package includes implementation of the central auctioneer agents (SMAA and RMAA) which, in turn, are wrappers for the MACE algorithm in package *edu*.

Package *org.catnets.optorsim.negotiations*. This is one of the most important packages of the simulator. It includes classes which implement economic agents (CSAs, BSAs and RAs), as well as the functions of the P2P mediators. Classes are included for all the types of messages exchanged during negotiations, when either the catallactic or centralised mechanisms are used. The classes in this packages are better described in sections 3.2.4 and 3.2.5, where the integration of the two market into the simulator is described.

Package *org.catnets.optorsim.time*. This package implements the simulated time. Different time model are available, one time based and one event driven. When the first model is adopted, simulation proceed in real time. When the event driven model is adopted, simulations speeds up because, when all threads are idle, the simulator is able to advance simulation time to the point when the next thread should be activated.

Package *org.catnets.optorsim.utils*. Some utility functions are implemented by classes in this package, including a function for the recording of simulation metrics, described in Section 3.2.6.

3.2.4 Integration of centralised market

The following sections describe the integration of the auctioneers into OptorSim. This includes a double auction implementation for the service market and an implementation of a multi-attribute combinatorial exchange for the resource market. The conceptual design of both mechanisms is described in Deliverable D1.1 [WP105]; the implementation of the auctioneer components is outlined in Deliverable D1.2 [WP106].

The section is structured as follows: Section 3.2.4 outlines the integration of the double auction component into the service market. Section 3.2.4 describes the integration of the multi-attribute auction into the resource market.

Service Market

In the service market, a double auction instance is applied for coordinating service allocations. In a double auction market [Fri91], a large number of agents trade a common object and can submit bids to a central auctioneer. Trading in double auctions is organized by means of order books, each for a set of homogeneous goods. In the CATNETS scenario there will be n different order books, each for one of the n different basic services. For a detailed description of the auction mechanism, we refer to [SNVW06] and Deliverable D1.1 [WP105].

In OptorSim, the service market auctioneer is represented as an agent. This auctioneer gets instantiated by the simulator during its initialization and can be contacted by every other agent. Complex service agents and basic service agents communicate with the auctioneer by means of messages, i.e. they can submit their bids in form of messages.

Furthermore, they can receive further information from the auctioneer agent such as the current market price. In case the auctioneer cleared the market – i.e., it computed an outcome and prices – agents get informed whether or not they are part of the allocation.

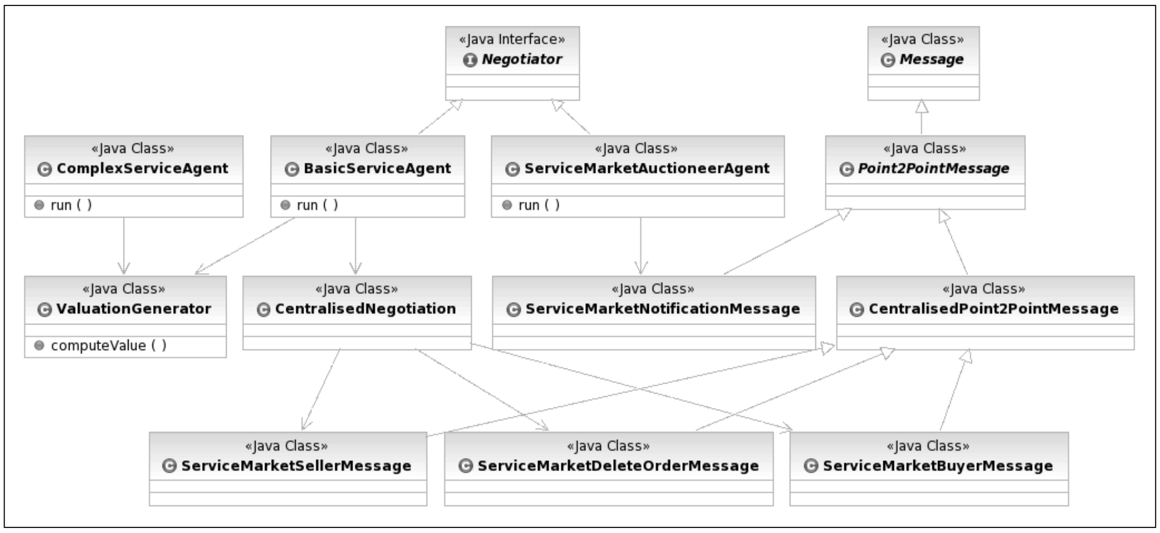

«Java Interface» Negotiator
«Java Class» Message
«Java Class» ComplexServiceAgent
run ( )
«Java Class» BasicServiceAgent
run ( )
«Java Class» ServiceMarketAuctioneerAgent
run ( )
«Java Class» Point2PointMessage
«Java Class» ValuationGenerator
computeValue ( )
«Java Class» CentralisedNegotiation
«Java Class» ServiceMarketNotificationMessage
«Java Class» CentralisedPoint2PointMessage
«Java Class» ServiceMarketSellerMessage
«Java Class» ServiceMarketDeleteOrderMessage
«Java Class» ServiceMarketBuyerMessage

Figure 3.21: Service market integration

Figure 3.21 briefly outlines the basic classes concerning the service market integration. At heart of the central service market stands three classes: `ComplexServiceAgent`, `BasicServiceAgent`, and `ServiceMarketAuctioneerAgent`. First, the `ComplexServiceAgent` class represents a buyer in the service market. Each complex service agent is represented by an individual instance of this class. Second, the `BasicServiceAgent` class denotes a seller in the service market. Likewise the complex service, each individual basic service agent is represented by an instance of this class. Last, the `ServiceMarketAuctioneerAgent` implements the central auctioneer as an agent. This auctioneer has access to the auction implementation, i.e. it can forward bids to the order book, it can trigger allocation decisions, and it can receive further information from the auction implementation (c.f. Deliverable D1.2 [WP106]). All three agent classes implement the `Negotiator` interface, i.e. these classes can communicate with each other by means of the methods declared by this interface.

A complex and a basic service agent each instantiates a `CentralisedNegotiation` class which is responsible for submitting bids to the auctioneer. In case an agent wants to trade, it first generates a valuation or reservation price for the service. This means, it determines the value of the bid which it would like to submit. This is realized by the `ValuationGeneration` class which computes a value for the bid on the basis of market information and historical data (c.f. Deliverable D1.2 [WP106]). After that, the agent communicates this price to its `CentralisedNegotiation` instance. Subsequently, the negotiation object submits a bid to the auctioneer.

Complex and basic service agents communicate with the auctioneer by means of messages. For the central case, the most important message type is the `Point2PointMessage` which enables the communication between two agents. A child of the `Point2PointMessage` class is the `CentralisedPoint2PointMessage` class which directly sends messages to the auctioneer. For the service market, the following messages are exchanged between the agents:

`ServiceMarketBuyerMessage`: A buyer message is sent, whenever a complex service agent wants to buy a service. The message contains the type of the required service and the agent's valuation (maximum price). Furthermore, agent specific information such as its ID and its location are stored in this message. The message is directly sent to the auctioneer.

`ServiceMarketSellerMessage`: A seller message is sent, whenever a basic service wants to sell a service on the service market. In analogy to the buyer message, this message contains information about the transaction object (which service), the agent's reservation price (minimum price), as as well as an ID of the agent.

`ServiceMarketDeleteMessage`: In case an agent wants to delete a submitted order, it can send a delete message to the auctioneer. The auctioneer removes the corresponding order from the order book. Such a message is important if the negotiation session of the agent is terminated, e.g. due to a time-out. In such a case, it has to be ensured that all orders from this agent are removed from the order books. The message contains all relevant information such as the agent's ID and the ID of its submitted order.

`ServiceMarketNotificationMessage`: A market notification message is sent if the auctioneer has computed an outcome. Successful agents – i.e. agents that are part of the allocation – get messages including their counterpart agent and the transaction price. In case of a call market[2], the unsuccessful agents also receive such a message including a negative price ($p = -1$). Besides distributing allocation decision, this type of message is also used whenever an agent finished a job. Suppose a complex service executes a job for which it uses a basic service. If the job is finished, the complex service informs the corresponding basic service that the execution is completed. Subsequently, the basic service agent can *unlock* this service and sell it again on the market. It is to note that the content of these messages may vary, e.g. the content informs that the allocation was successful or that a job finished. Agents can interpret the context of such messages.

An example of the interaction between the agents and the auctioneer is outlined in figure 3.22. A basic service agent wants to sell a basic service on the market and

[2]See Deliverable D1.1 [WP105] for a detailed definition.

has already generated valuation for the bid. For this, it calls the `doNegotiation` method of its `CentralisedNegotiation` instance (1). This instance generates a new `ServiceMarketSellerMessage` and submits this message to the auctioneer (2). Likewise, a complex service wants to acquire a basic service. The service calls the `doNegotiation` method of its `CentralisedNegotiation` instance (3). This instance submits a `ServiceMarketBuyerMessage` to the auctioneer (4). Having received a set of messages, the auctioneer computes an outcome.[3] In the example, both agents are successful, i.e. the valuation of the complex service agent is greater than the reservation price of the basic service agent. Thus, the auctioneer distributes two `ServiceMarketNotificationMessage` (5, 6) which includes all relevant information that are required by both counterparts. The basic service starts to buy the required resources on the resource market. It informs the complex service with a `ServiceMarketNotificationMessage` (7) if all resources are allocated. After that, the complex service starts to execute its job. If the job is finished, it informs the basic service by another `ServiceMarketNotificationMessage` to free the service.

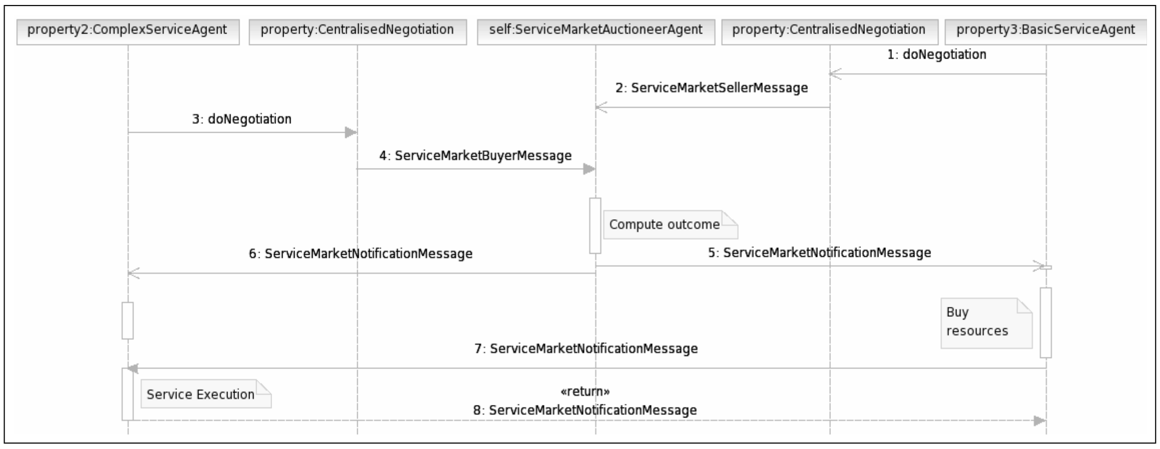

property2:ComplexServiceAgent
property:CentralisedNegotiation
self:ServiceMarketAuctioneerAgent
property:CentralisedNegotiation
property3:BasicServiceAgent
1: doNegotiation
2: ServiceMarketSellerMessage
3: doNegotiation
4: ServiceMarketBuyerMessage
Compute outcome
6: ServiceMarketNotificationMessage
5: ServiceMarketNotificationMessage
Buy resources
7: ServiceMarketNotificationMessage
Service Execution
«return»
8: ServiceMarketNotificationMessage

Figure 3.22: Service market sequence

Resource Market

For allocating services in the resource market, we apply a multi-attribute combinatorial exchange (MACE) [SNVW06]. Basic service agents and resource service agents submit their bids to the auctioneer instance. After that, the bids are transformed into an internal representation form and, subsequently, the winners are determined (allocation). Finally, prices are computed in consideration of the allocation. As a result of the market mechanism, agents get informed whether or not they are part of the allocation. A description of

[3]A key consideration in exchanges is the timing of the clearing process, i.e. the timing of determining the auction winners. Exchanges can be either cleared continuously (Continuous Double Auction) or periodically (Periodic Double Auction, Call Market). For a description of these different clearing strategies, we refer to Deliverable D1.1 [WP105].

the implementation of the resource market component can be found in Deliverable D1.2 [WP106].

The resource market is integrated similarly into OptorSim as the service market. The auctioneer is represented as a `ResourceMarketAuctioneerAgent` and has access to the market implementation. Instances of the `BasicServiceAgent` classes act as buyers on this market. In analogy to the service implementation, they generate a valuation by means of the `ValuationGeneration` class and communicate their requirements to their `CentralisedNegotiation` instance. Likewise, an instance of the `ResourceAgent` acts as a seller of resources. It has an instance of the `ValuationGeneration` class and of the `CentralisedNegotiation` class.

The communication between agents is realized by means of messages. The resource market provides similar message types as the service market:

- a `ResourceMarketBuyerMessage` for submitting buyer bids,
- a `ResourceMarketSellerMessage` for submitting seller bids,
- a `ResourceMarketDeleteMessage` for deleting bids, and
- a `ResourceMarketNotificationMessage` for distributing notifications.

Whenever a basic service agent has sold a service on the service market, it starts to bid for the required resources on the resource market. In case it gets all required resources allocated, it notifies the complex service that its job can now be executed. This process is depicted in figure 3.23: A resource agent has free capacity[4] and submits a seller order to the auctioneer (1, 2). Furthermore, a basic service allocated a service on the service market and requires resources for executing it. As such, it submits an order to the resource market (3, 4). The auctioneer computes an outcome and informs the participating agents about the allocation decision. In this example, both agents are part of the allocation. By means of a `ResourceMarketNotificationMessage` (5), the resource agent gets to know that its resources are required to execute a basic service. The agent pins it, i.e. it marks the resources as locked. When the basic service agent receives its `ResourceMarketNotificationMessage` (6), it notifies the complex service that the job can be executed. The basic service agent waits until the complex service finished the job, i.e. until it gets a message from it. After that, it informs the resource agent, that it can unlock its resources (7).

If the basic service agent cannot allocate the required resources on the market, it notifies the complex service that it cannot provide the service. This is also realized by sending `ResourceMarketNotificationMessage`. In this case, the content of this message describes a *failed allocation*.

[4]This means that it has free resources that it can sell on the market.

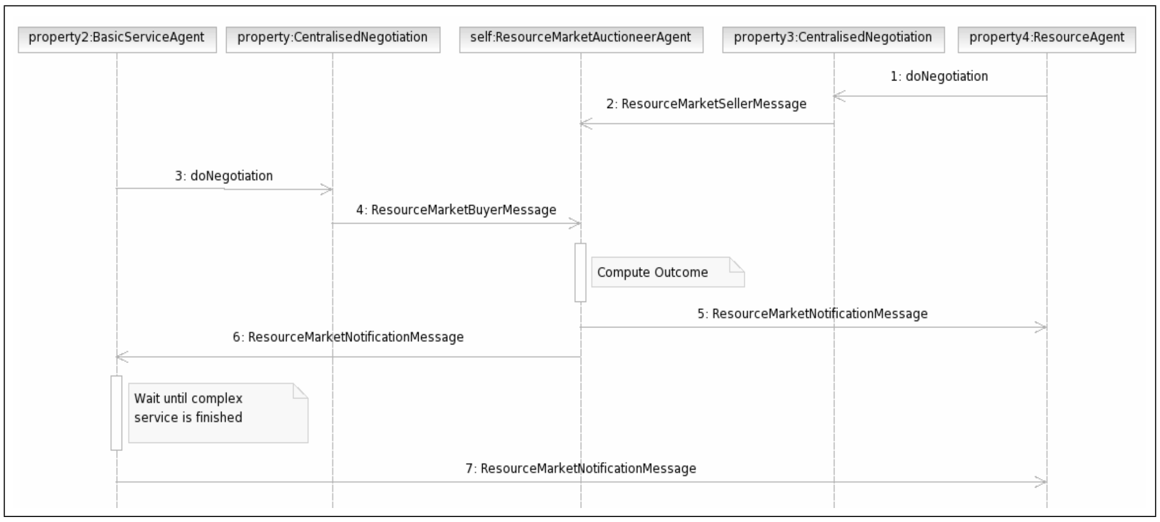

property2:BasicServiceAgent
property:CentralisedNegotiation
self:ResourceMarketAuctioneerAgent
property3:CentralisedNegotiation
property4:ResourceAgent
1: doNegotiation
2: ResourceMarketSellerMessage
3: doNegotiation
4: ResourceMarketBuyerMessage
Compute Outcome
5: ResourceMarketNotificationMessage
6: ResourceMarketNotificationMessage
Wait until complex service is finished
7: ResourceMarketNotificationMessage

Figure 3.23: Resource market sequence

3.2.5 Integration of catallactic market

This section describes the integration of the catallactic agents into OptorSim. This includes the bilateral bargaining protocol implementation for the service and the resource market. The conceptual design of the mechanism is described in Section 4.2.2 of Deliverable D1.2 [WP106]; the implementation of the strategy core is presented in Deliverable D1.1 [WP105].

In OptorSim, the service and the resource markets are both realised with software agents. Three agent types were implemented: the complex service agent, the basic service agent and the resource agent. They communicate with each other by exchange messages using the P2P system of OptorSim. During the initialisation of the simulator, they get instantiated and wait for incoming messages. They message are passed to the defined strategy module of every software agent. A complex service agent uses a buyer catallactic reasoner enabling him to buy basic services. The basic service agent instantiates two catallactic reasoners, on for selling his service and one for buying resource bundles. Similar to the complex service agent, the resource agent uses only one reasoner for selling its resource bundles.

Major difference between the integration of the agents in the centralised and catallactic case lies in the reactive and proactive behaviour, that OptorSim provides for the agent development. For the integration of the catallactic agents, the behaviour is exclusively reactive, whereas the agents in the central case is proactive by sending messages to the central auctioneer.

Figure 3.24 outlines the most important classes concerning the integration of the service and resource markets. Three classes are essential for the implementation of the software agents: `ComplexServiceAgent`, `BasicServiceAgent`, and

`ResourceAgent`. First, the `ComplexServiceAgent` represents a buyer in the service market. Each complex service agent is represented by an individual instance of this class. Second, the `BasicServiceAgent` class denotes a seller in the service market. Likewise the complex service, each individual basic service agent is represented by an instance of this class. Last, the `ResourceAgent` implements a seller of resource bundles. This implementation of a resource agent is able to manage the resources which means the resource agent allocates the resources and frees after consumption.

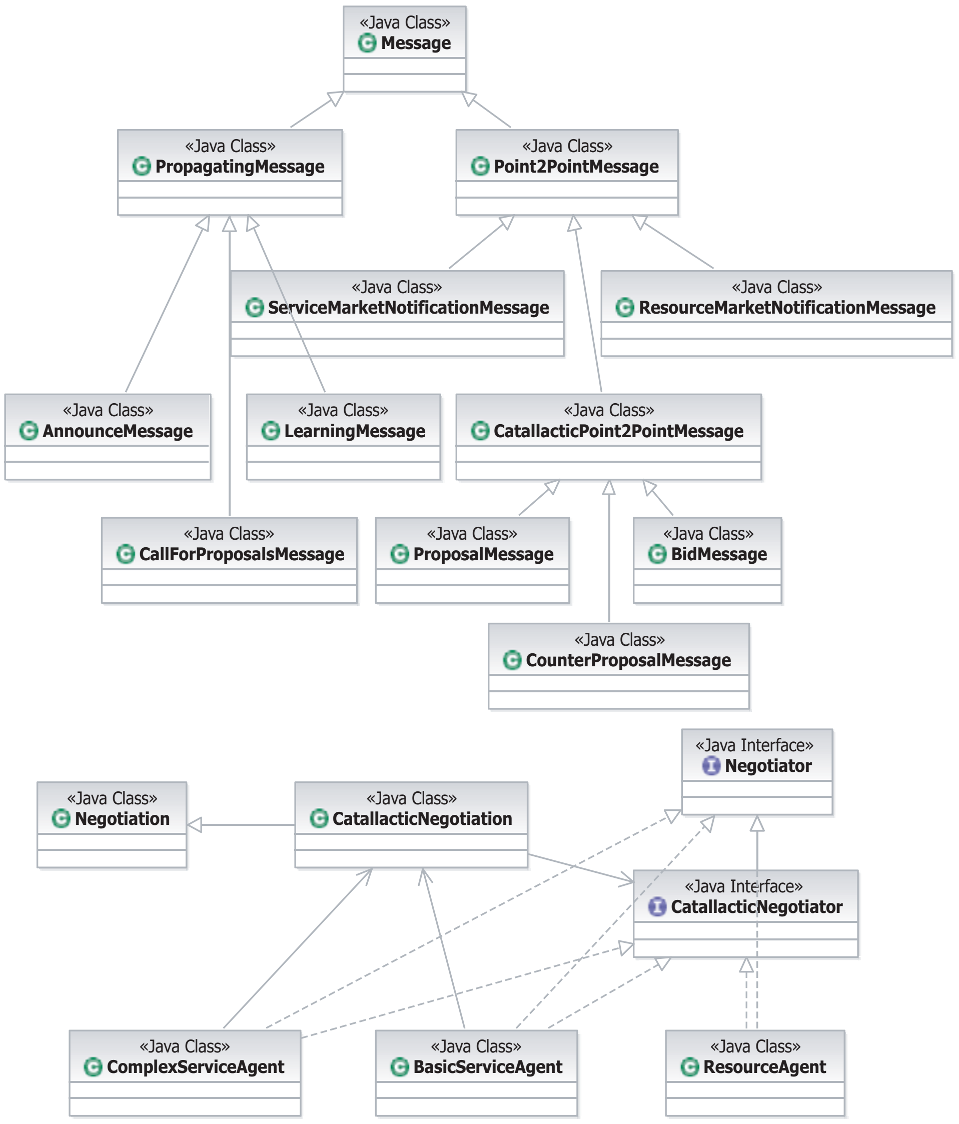

«Java Class» Message
«Java Class» PropagatingMessage
«Java Class» Point2PointMessage
«Java Class» ServiceMarketNotificationMessage
«Java Class» ResourceMarketNotificationMessage
«Java Class» AnnounceMessage
«Java Class» LearningMessage
«Java Class» CatallacticPoint2PointMessage
«Java Class» CallForProposalsMessage
«Java Class» ProposalMessage
«Java Class» BidMessage
«Java Class» CounterProposalMessage
«Java Interface» Negotiator
«Java Class» Negotiation
«Java Class» CatallacticNegotiation
«Java Interface» CatallacticNegotiator
«Java Class» ComplexServiceAgent
«Java Class» BasicServiceAgent
«Java Class» ResourceAgent

Figure 3.24: Service and resource market integration.

All software agent implementations are threads and implement the `CatallacticNegotiator` interface, which enables the access to the catallactic reasoner implementation.

Starting and running a bilateral negotiation. In OptorSim, a negotiation is started using with the instantiation of the `Negotiation` class and the call of the `doNegotiation` method. This class has been extended to the `CatallacticNegotiation` class fulfilling the requirements of the bilateral negotiation. New methods have been introduced to separate the service (method `doServiceNegotiation`) and resource market negotiation (method `doResourceNegotiation`). A call of these methods starts a service or resource negotiation, respectively. The caller will get back the result of the ended negotiation. This is the complex service agent for a service market negotiation and the basic service agent on the resource market.

As already mentioned, messages are used for communication between the software agents. For the bilateral negotiation protocol in the catallactic case, several messages have been introduced. The most important message types are the `Point2PointMessage` and the `PropagatingMessage`. The `Point2PointMessage` class realizes a one-to-one communication between to agents, where as the `PropagatingMessage` class implements a broadcast to all agents. Every agent filters this broadcast message itself, implementing simple filter rules. Finally, the transmission of the messages is performed by passing the message to the `P2PMediator` which provides access to the underlying peer-to-peer infrastructure connecting the nodes.

For integration of communication for the service and the resource market, several specialized message subclasses of the two base communication classes have been created. The following messages are exchanged between the agents (see Figure 3.24):

`CallForProposalMessage:` This message broadcasts a new request for a service instance or resource bundle to all local and remote agents. This message starts the negotiation by discovering possible negotiation partners. A receiver of this message checks whether he is able to provide the asked service type or not. If he is willing to offer this service, a `BidMessage` is created and sent back to the requester.

`AnnounceMessage:` In general, this message class is a generic broadcast implementation. Currently, this message broadcasts the selected negotiation partner after a price ranking. Any receiver, which sent a `BidMessage` handles this message.

`LearningMessage:` This message broadcasts the fitness information to all agents which trade the specified good. Learning messages are collected during a running negotiation and interpreted after a negotiation ended.

`ProposalMessage:` A proposal message is a message from a buyer to a seller suggesting a new price for the traded good, accepting or rejecting it.

`CounterProposalMessage:` A counter-proposal message is the answer message to a proposal message. This message is sent from a seller to a buyer.

`BidMessage:` A bid message returns a bid related to call-for-proposal message.

`ServiceMarketNotificationMessage`: This message notifies the basic service agent about the end of a negotiation on the service market.

`ResourceMarketNotificationMessage`: This message notifies the resource agent about the end of a negotiation. The resource agent will free the reserved resources for the specified request.

An example of the interaction between a complex service agent and a basic service agent shows figure 3.25. A complex service wants to buy a basic service on the market. The agent instantiates a new `CatallacticNegotation` and calls the `doServiceNegotiation` method of this class. This instance creates a `CallForProposalMessage` and broadcasts this message to all basic service agents. A basic service agent receives the message and generates a `BidMessage` offering the asked service. Having received a set of possible negotiation partners, the complex service agent ranks the offers and chooses the cheapest offer to start negotiating with. First, a `AnnounceMessage` is created. This message broadcasts the winner of the ranking. Second, a `CounterProposalMessage` is sent to the winner basic service. The basic service agent answers with a `ProposalMessage` containing an accept. The complex service agents receives the result of the bilateral negotiation. In the current implementation, the results contains the identifier of the contracted basic service.

Similar to the service negotiation a resource negotiation is started by the contracted basic service agent. The basic service agent uses a predefined, static mapping table to create a `CallForProposalMessage`. At the end of the resource market negotiation the `ServiceMarketNotificationMessage` signals either a successful or failed negotiation to the `ComplexServiceAgent`.

Using this bilateral negotiation protocol, several metrics are defined and measured in the simulator. The next section shows the current implemented metrics.

3.2.6 Metrics

Deliverable D4.1 [WP405] describes a set of technical and economical metrics for the evaluation of the central and calallactic allocation mechanisms.

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

For this reason in the second year of the project a subset of the envisioned metrics framework has been implmented into the CATNETS simulator. The set of implemented

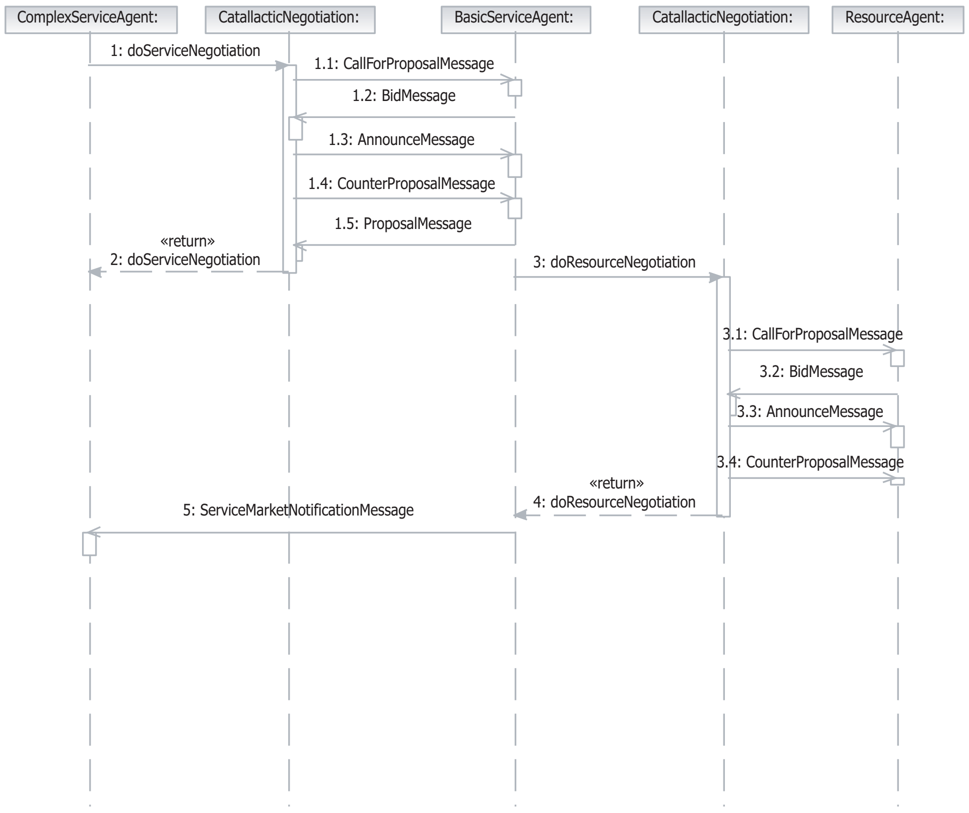


Figure 3.25: Bilateral negotiation on the service and resource market.

metrics has also been chosen by taking in account the metrics which are recorded in the prototype and an effort has been made in order to minimise the differences.

The implementation of the metrics is independent from the (centralised or catallactic) economic model. A description of the implmented metrics is given in Section 2.3 of Deliverable D4.2 [WP406].

Chapter 4

Simulation of ALNs

The second task in the workplan of WP2 for the second year of the project was the “Simulation of application layer networks and refinement”. This chapter illustrates the work done for this task. In particular, the chapter includes:

- an handbook of the CATNETS simulator;
- the description of some functional tests performed to validate the manual scenario generator and the simulator.

4.1 Simulator handbook

The code of the CATNETS simulator is stored into a CVS repository at the University of Bayreuth and we currently run it within the Eclipse SDK [ecl] synchronised with the repository.

There are five configuration files used to control various inputs to the CATNETS simulator. Four files are created by the *Scenario Generator* described in Section 3.1. These are:

`topology.conf` The ALN Configuration File describes the ALN topology and the content of each site; that is, the resources available and the network connections to other sites.

`bs.conf` The Basic Service Configuration File associates each basic service to the resource bundle needed for its execution;

`arb.conf` The Available Resource Bundle configuration files defines the resources which are part of bundles provided by RAs;

`cs.conf` The Complex service Configuration File contains information on the simulated CSs, and the site policies for each site (the list of CS each site will accept).

The 5th configuration file is the Simulation Parameters File `examples/parameters_catnets.conf` which contains various simulation parameters which the user can modify.

Sample configuration files for the CATNETS simulator can be found in the `examples/` directory.

The main class is *org.catnets.optorsim.OptorSimMain*. This classes uses the default parameters file located at `examples/parameters_catnets.conf`.

4.1.1 The ALN Configuration File

The file describes the status of the resources of each site and the layout of the simulated ALN. The example configuration file shown in Figure 4.2 describes the ALN in Figure 4.1.

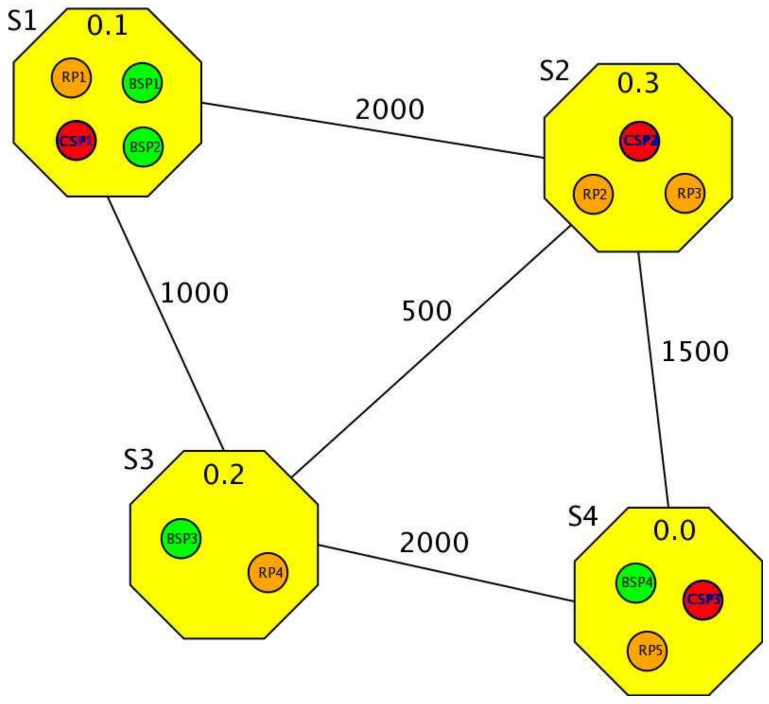


Figure 4.1: An example ALN.

Each row in the configuration file is information about one site:

- the first column shows the failure probability of the site.

```
# Nodes configuration for Catnets test - Catallaxy
#
# failure prob., #CSAs, #BSAs, #RAs, SMAA, RMAA,
# bs ids, arbs ids, links with bandwidth
#
0.1 1 2 1 F F bs1 bs2 arb1 0. 2000. 0. 1000.
0.3 1 0 2 F F arb2 arb3 2000. 0. 1500. 500.
0.0 1 1 1 F F bs4 arb5 0. 1500. 0. 2000.
0.2 0 1 1 F F bs3 arb4 1000. 500. 2000.
```

Figure 4.2: An example ALN configuration file (Catallaxy).

- the second column is the number of Complex Service Agents at the site.
- the third column is the number of Basic Service Agents at the site.
- the fourth column is the number of Resource Agents at the site.
- the fifth and sixth columns are two boolean values which specify if the SMAA and RMAA, respectively, are located on the site.
- the following identifiers specify the Basic Services provided by the BSAs at the Available Resource Bundles provided by the RAs at the site.
- The rest of the table is a site vs. site matrix giving the bandwidth (i.e. link capacity) between each site in Mb/s. The matrix is diagonally symmetric. Entries along the diagonal are ignored since network bandwidth within a site is assumed to be infinite.

When a simulation of the centralised allocation mechanism is performed, the file has the form presented in Figure 4.3, where an additional site for the central auctioneers, fully connected to all other sites, is also specified.

4.1.2 The Basic Service Configuration File

This file includes identifies of the basic service provided by BSAs. An example is given in Figure 4.4.

Each row in the configuration file is the information for one basic service:

- the first column is the identifier of the basic service;
- the second column is the name of the basic service;
- the third column is the quality of the basic service;
- the following values specifies the resource bundle needed for service execution in terms of $\langle resource\_name, quantity\rangle$ pairs.

```
# Nodes configuration for Catnets test - Centralised
#
# failure prob., #CSPs, #BSPs, #RPs, SCA, RCA,
# bs ids, arbs ids, links with bandwidth
#
0.1 1 2 1 F F bs1 bs2 arb1 0. 2000. 0. 1000. 1000.
0.3 1 0 2 F F arb2 arb3 2000. 0. 1500. 500. 1000.
0.0 1 1 1 F F bs4 arb5 0. 1500. 0. 2000. 1000.
0.2 0 1 1 F F bs3 arb4 1000. 500. 2000. 1000.
#
# Site for Central Auctioneers
#
0.0 0 0 0 T T 1000. 1000. 1000. 1000. 0.
```

Figure 4.3: An example ALN configuration file (Centralised).

```
# Basic service configuration for Catnets test
#
bs1 pdf_converter gold cpu 70 storage 40
bs2 printer_service silver cpu 30 storage 10 ram 50
bs3 pdf_converter silver cpu 50 storage 20
bs4 printer_service silver cpu 30 storage 10 ram 50
```

Figure 4.4: An example Basic Service configuration file.

4.1.3 The Available Resource Bundle Configuration File

The file includes identifiers of the available resource bundles provided by RAs. An example is given in Figure 4.5.

```
# Available resource bundle configuration for Catnets test
#
arb1 cpu 100 storage 40
arb2 cpu 100 storage 40 ram 150
arb3 storage 70 ram 100
arb4 cpu 50
arb5 cpu 100 storage 40
```

Figure 4.5: An example Available Resource Bundle configuration file.

Each row in the configuration file is the information for one available resource bundle:

- the first column is the identifier of the available resource bundle;

- the following values specifies the resources available in the bundle in terms of $\langle resource\_name, quantity \rangle$ pairs.

4.1.4 The Complex Service Configuration File

An example file is shown in Figure 4.6.

```
#
# CS Table
#
# A CS name and a list of basic service needed.
#
\begin{cstable}
cs1 bs1 bs2 bs3
cs2 bs3 bs4
cs3 bs2
cs4 bs1 bs2 bs3 bs4
\end{cstable}
#
# CSP Schedule Table
# CSP site id, CS it will run
#
\begin{cspscheduletable}
0 cs1 cs2 cs3 cs4
1 cs1 cs2 cs3 cs4
2 cs1 cs2 cs3 cs4
3 cs1 cs2 cs3 cs4
\end{cspscheduletable}
#
# The probability each cs runs
#
\begin{csselectionprobability}
cs1 0.25
cs2 0.25
cs3 0.25
cs4 0.25
\end{csselectionprobability}
```

Figure 4.6: An example CS configuration file.

It is structured as follows:

- the first part (between `begin{cstable}` and `endcstable` defined the set of runnable complex services. Every row starts with a CS identifier followed by the list of basic services which by which the CS is composed.

- each row in the second part of the CS configuration file (between `begin{cspscheduletable}` and `end{cspscheduletable}`) gives the scheduling policy for each site i.e., which CSs the CSAs located on a site are willing to run. In this example all CSAs on all sites will be able to run all the defined CSs.
- the third part of the CS configuration file, which is between `begin{csselectionprobability}` and `end{csselectionprobability}`, determines the frequency with which each CS will be submitted to the ALN. In the example above all the four CS are given the same probability of being submitted.

4.1.5 Simulation Parameters

The simulation parameters are set manually by the user in a parameters file. The default parameters file is found in `examples/parameters_catnets.conf`. Following is an explanation of each parameter.

General Parameters

aln.topology.file - The configuration file to describe the ALN topology.

aln.bs.file - The configuration file to describe the basic services.

aln.arb.file - The configuration file to describe the available resource bundles.

cs.configuration.file - The configuration file to describe the complex services.

number.complexservices - The number of CSs submitted during the simulation run.

users - Determines the pattern in which ALN users submit CSs to the Complex Service Dispatcher. Options:

1. **Simple:** submit CSs at regular intervals until all CSs have been submitted. The interval is set by the **cs.delay** parameter (below).
2. **Random:** submit CSs at intervals which are uniformly random between zero and twice the **cs.delay**.

policy - Determines the scheduling policy of the Complex Service Dispatcher. Options:

1. **Random:** CSs are scheduled randomly to any CSA that will run the CS.

2. **Queue Length:** schedules to the CSA with the shortest queue of waiting CSs. If two CSAs have the same shortest queue length one of them is chosen at random.

cs.delay - The basic time interval (in milliseconds) between CSs being submitted to the ALN by the Users during simulation. The actual submission interval depends on the type of user chosen (above).

access.pattern.generator - Determines the order in which BSs are accessed within a CS. Options:

1. **SequentialAccessGenerator:** CSs are accessed in the order stated in the CS configuration file.
2. **RandomAccessGenerator:** CSs are accessed using a uniform random distribution;
3. **RandomWalkUnitaryAccessGenerator:** CSs are accessed using a unitary random walk, starting from a CS chosen using a uniform random distribution.
4. **RandomWalkGaussianAccessGenerator:** CSs are accessed using a Gaussian random walk, starting from a CS chosen using a uniform random distribution.
5. **RandomZipfAccessGenerator:** CSS are accessed using a Zipf-like distribution[1].

shape - The shape parameter α for the Zipf-like access pattern.

random.seed - Determines whether the seed used by various methods within the CATNETS simulator where random numbers are required is fixed or random. If this is set to `yes`, it will be random; if `no`, it will be fixed. For example, if it is `yes`, a different set of CSs will run each time the simulation is run. If it is `no`, the same CSs will run each time.

max.queue.size - The maximum number of CSs the CSA will hold in its queue before it refuses to accept any more.

bs.execution.time - The time in milliseconds for a CSA to execute each BS.

Market Parameters

These parameters are specific for the service/resource allocation mechanisms.

market.model - Set to `1` to use the catallactic allocation mechanism or `2` to use the centralised mechanism.

[1]In a Zipf-like distribution the popularity of the BSs associated to a CS is given by $P(f_i) = 1/i^\alpha$, with $0 \leq \alpha < 1$, where $P(f_i)$ is the popularity of the i-th most popular CS and α is close to 1.

market.central.service.clear - Clearing policy for the centralised service market: `1` Call Market or `2` Continuous.

market.central.service.clearinterval - Call market clearing interval for the centralised service market; defines after how many ms the market will be cleared.

market.central.resource.clear - Clearing policy for the centralised Resource Market: `1` Call Market or `2` Continuous

market.central.resource.clearinterval - Call market clearing interval for the centralised resource market; defines after how many ms the market will be cleared.

market.decentralized.sm.file - File containing initialisation parameters for the catallactic allocation strategy on the service market.

market.decentralized.rm.file - File containing initialisation parameters for the catallactic allocation strategy on the resource market.

Negotiation Parameters

These parameters regulate how negotiations are conducted in both the catallactic and centralised mechanisms.

timeout - The time (in milliseconds) an agent will wait for non-blocking reception of messages during negotiations for services or resources.

hop.count - Regulates the propagation of broadcast messages over the network when the catallactic mechanisms is adopted

Other parameters

time.advance - There are two time models implemented in the CATNETS simulator one time-based and one event-driven, and the simulator can be run in either mode with the same end results. In the time-based model, the simulation proceeds in real time. In the event-driven model, whenever all the threads are inactive, the simulation time is advanced to the point when the next thread should be activated. The use of the event-driven model speeds up the running of the simulation considerably, whereas the time-based model may be desirable for demonstration or other purposes.

time advance - Set to `yes` to use the event-driven time model or `no` to use the time-based model.

The time-bases model is the default.

metrics.path - The path where files recording metrics collected during simulations are stored.

4.2 Simulator functional test

4.2.1 Validation of the scenario generator

The manual scenario generator has been used to create a medium-scale scenario to be used for the evaluation of the centralised allocation mechanism. We decided to generate an ALN having 20 nodes all connected to the special site where the two auctioneers are located. The complete set of parameters defining the scenario is shown in Table 4.1.

# of ALN nodes	21
node failure probability	0 for each node
Bandwidth	1000 for each edge
# of CSAs	5 (max 1 for node)
# of BSAs	30 (max 4 for node)
# of RAs	15 (max 3 for node)
# of resources	5
Resource quantity	value between 20 to 50
# of basic services	5
BS quality	{platinum, gold, silver, bronze}
Resource bundle cardinality	between 1 and 5
# of complex cervices	5
CS execution probability	0.2

Table 4.1: Parameters for Scenario Generator Functional Test.

The generated configuration files are included in the `examples/` directory. The generation process took about two hours. Even though the generation was done by the developer of the scenario generator, we believe that two hours is a reasonable time for the definition of a scenario of this size. However, it is clear that the manual generator cannot be used for the generation of larger scenarios, for which the use automatic generator is necessary.

4.2.2 Validation of simulator

For a proof-of-concept evaluation of the simulator, we have run several simulation runs to test the integration of the auctions (central case). In Deliverable D1.2 [WP106], we discuss some of the preliminary results from these tests. The results are neither meant to be convincing nor to be statistically evident.

We have run 10 different simulation runs using a small configuration file with 3 agents and 2 different resources. We measured a subset of the metrics defined in WP4, as explained in Deliverable D4.2 [WP406]. As a main result of these test runs, we identified that the total number of complex service requests is greater than the number of allocated ones. In general, this ratio was low during most of the simulations runs. In order to improve this ratio, we have to balance the valuation generation of the agents (cf. [WP106]) as well as the strategies of the bidding agents in project year 3.

Chapter 5

Relations to other WPs

The work done by WP2 is strictly related to the activities performed by the other work packages of the CATNETS project. This chapter outlined the principal relationships between WP2 and the other WPs.

5.1 WP1

The objective of WP1 (Theoretical and Computational Basis) is the conceptual design of market mechanisms for the CATNETS scenario. This includes the design of auctions (denoted as central case) and a decentralized bargaining strategy (denoted as Catallactic case). The relations to WP2 are the following ones:

- The central auctions and the decentralized bargaining strategies are integrated into the simulator. This includes a deep collaboration concerning the simulation model and the message system.
- The development of the software patterns and components that are required to simulate market based ALNs has been conducted in tight cooperation between WP1 and WP2.

5.2 WP3

The objective of WP3 is to have a proof-of-concept application that is used to demonstrate the ideas being developed, among others, in WP1. The relations to WP3 are as follows:

- The simulator embeds into a single framework both the central and catallactic allocation mechanisms while the prototype embeds only the catallactic mechanism.

The reason for this is that the prototype is supposed to demonstrate the feasibility of real-world ALN applications based on the catallactic service/resource allocation mechanism, while the goal of the simulator development is to produce a tool which permits to run a large set of scenarios for an effective and extensive comparison of the performance of the two allocation mechanisms.

- As the CATNETS simulator directly derive from the grid simulator OptorSim, the implementation of the agent communication layer differs from the implementation of the corresponding middleware in the prototype. This implies some differences in the way the technical metrics are collected and how the catallactic strategy is implemented in the two tools. However, an effort has been made in order to minimise these differences so that the results obtained by the two are potentially comparable. In the third year of the project an investigation of their actual comparability has to be performed. As a precondition, a set of simulation scenarios reflecting the operation condition of the prototype application has to be defined.

5.3 WP4

The first goal of WP4 is to evaluate the performance of the Catallactic approach by means of the simulator described in this deliverable and the prototype developed by WP3 (see deliverable D3.1 [WP305] and D3.2 [WP306]). The second goal is to compare, using the simulator, the performace of the catallactic and centralised mechanisms. The relations to WP4 are as follows:

- The identification and formalization of relevant metrics to compare centralised and catallactic market mechanisms has been done in cooperation between WP4 and WP2.
- Analysis of which metrics can be measured by both the CATNETS simulator and the prototype has been done in cooperation among WP4, WP3, and WP2.

Chapter 6

Conclusions

6.1 Achieved results

Two tasks were in the workplan of WP2 - Simulation in project year two: “Simulator enhancement to support new architecture properties” and ”Simulation of ALNs and refinement”.

Simulator enhancement to support new architecture properties. The work done for this task can be summarised as follows:

- definition of an abstract model for ALNs to be simulated.
- development of a manual ALN scenario generator, to be used for the definition of simulation configurations.
- extension of the Grid simulator OptorSim in order to embed: (1) the code implementing the central auctioneers; (2) the code implementing the catallactic bargain strategy; (3) a flexible messaging system to support both centralised and catallactic allocation mechanisms; (4) a partial implementation of the functions for recording the metrics defined by WP4; an effort has been made in order to minimise the differences with the metrics implemented in the prototype by WP3.

Simulation of ALNs and refinement. Two activities have been performed:

- Definition of a simulator handbook describing the input files which defines the scenario to be simulated.
- Preliminary experiments for validation of the manual scenario generator and the simulator.

6.2 Future work

According to the workplan of the project, in year 3 WP2 is supposed to continue working on task "Simulation of Application Layer Networks and refinement". In particular, the work will be conducted along the following directions.

Automatic scenario generator. This tool is needed in in order to generate large-scale scenarios. Its requirements have already been defined and its implementation has started. The implementation is planned to be concluded by T0+28.

Full implementation of metrics. The metrics framework developed by WP4 is currently partially implemented into the CATNETS simulator. The full implementation of metrics is planned to be completed by T0+28.

Bandwidth dependent message delivery. The current version of the simulator is not able to simulate bandwidth dependent message delivery. This means that messages are delivered from, say, site S_1 to site S_2 in the ALN, regardless the capacity of the network links between the two sites. OptorSim embeds a mechanism which can be exploited to simulate this missing feature. This task is supposed to be completed by T0+30.

Extensive simulation. An extensive simulation of different ALN scenarios is necessary to compare performances of the centralised and catallactic allocation mechanisms. This tasks can start once the three activities above have been completed, i.e., at T0+30 and is planned to be completed by T0+34.

Simulation of prototype-like scenarios. This task is needed to effectively compare performance of the catallactic mechanism in the simulator and in the prototype. The task will start at T0+30 and is planned to be completed by T0+34.

Bibliography

[BCC$^+$03] W. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Optorsim - a grid simulator for studying dynamic data replication strategies. *Int. Journal of High Performance Computing Applications*, 17(4), 2003.

[CCSM$^+$04] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini. Analysis of Scheduling and Replica Optimisation Strategies for Data Grids using OptorSim. *Journal of Grid Computing*, 2, 2004.

[ecl] Eclipse web site. `http://www.eclipse.org`.

[edg] The DataGrid Project. `http://www.edg.org/`.

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

[opt] OptorSim web site. `http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html`.

[SNVW06] Björn Schnizler, Dirk Neumann, Daniel Veit, and Christof Weinhardt. Trading Grid Services – A Multi-attribute Combinatorial Approach. *European Journal of Operational Research, forthcoming*, 2006.

[WP105] WP1. Environmental Analysis of Application Layer Networks. Technical Report WP1 - D1, CATNETS EU IST-FP6-003769, 2005.

[WP106] WP1. Annual Report of WP1. Technical Report WP1 - D2, CATNETS EU IST-FP6-003769, 2006.

[WP205] WP2. Analysis of Simulation Environment. Technical Report WP2 - D1, CATNETS EU IST-FP6-003769, 2005.

[WP305] WP3. Analysis of Current Middleware used in Peer-to-Peer and Grid Implementations for Enhancement by Catallactic Mechanisms. Technical Report WP3 - D1, CATNETS EU IST-FP6-003769, 2005.

[WP306] WP3. Annual Report of WP3. Technical Report WP3 - D2, CATNETS EU IST-FP6-003769, 2006.

[WP405] WP4. Metrics Specification. Technical Report WP4 - D1, CATNETS EU IST-FP6-003769, 2005.

[WP406] WP4. Annual Report of WP4. Technical Report WP4 - D2, CATNETS EU IST-FP6-003769, 2006.

In this paper the simulation environment for the CATNETS project is defined further. The chosen simulator is adopted in terms of new features an architecture changes in order to provide a valid simulation environment for Application Layer Network scenarios. Furthermore the requirements for a scenario generator and the needed configuration mechanisms for the actual simulation runs are introduced.

ISSN 1864-9300