Ministerul Educaţiei al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Catedra Automatică şi Tehnologii Informaţionale

RAPORT

Disciplina: Programarea aplicațiilor distribuite

Tema: Procesarea datelor distribuite semi-structurate

A efectuat:

Vovc Artemie st. TI-133

A verificat:

Antohi Ionel lect. super. mag.

Chișinău 2016

Cuprins

1 Scopul lucrării

Scopul lucrării rezidă în studiul modelelor pentru procesarea datelor XML (DOM/ SAX) și JSON la distribuirea acestora.

Obiectiv: Dezvoltarea unui sistem cu date distribuite eterogene și utilizarea unui mediator pentru accesarea acestora. Crearea unui modul de validare a datelor de format XML ce parvin centralizat de la nodul de mediere, Maven, spre client.

Formularea problemei

Date legate (asociate) localizate diferit; vorbesc despre același lucru, dar diferă modelul, schema, convențiile…
Cum obținem datele?

- depozite date, *warehousing* - Centralizare date prin combinarea într-o schemă globală unică; reconstruire periodică.
- mediere, *mediation* - Crearea a unei vederi a informațiilor accesibile, readresarea interpelărilor spre sursele potrivite

În lucrarea curentă de laborator se va opta pentru modelul de mediere, în care rolul de mediator va fi preluat de nodul „maven”.

- protocolul de transport a datelor rămîne a fi TCP.
- nodurile informative vor transmite datele în format JSON, iar nodul Maven va transmite datele clientului în format XML.
- clientul la primirea datelor le va valida, utilizînd definiția documentului în XML Schema.

2 Sockets, UPD, TCP

Limbajul Java, prin setul de biblioteci API, pune la dispoziție suport nativ pentru aplicații în rețea. Cel mai simplu mod de a realiza comunicația între două mașini este prin utilizarea socket-urilor (soclurilor) de comunicație. Socket-urile sunt mecanisme de comunicație în rețea (dezvoltate la Universitatea Berckley din California) care pot folosi orice protocol de comunicație, deși, în general, se folosesc protocoalele din stiva TCP/IP.

Orice calculator conectat la Internet trebuie să aibă o adresă prin care să fie identificat, adresă care se numește adresă IP (Internet Protocol). O adresă IPv4 (versiunea 4) este formată din 4 octeți, iar reprezentarea ei externă se face printr-o secvență de 4 numere separate prin puncte (ex: 192.168.0.1). Pe lângă această reprezentare, se mai pot folosi nume simbolice grupate în domenii, iar corespondențele dintre adresele IP și numele simbolice sunt stocate într-o bază de date distribuită, accesată prin serviciul numelor de domenii (DNS – Domain Name Service).

În Java, clasele și interfețele de programare în rețea sunt cuprinse în pachetul java-net. O adresă IP se reprezintă prin clasa InetAddress care conține atât adresa IP (în format numeric) cât și adresa simbolică corespunzătoare. Această clasă nu are un constructor public, un obiect din această clasă se poate crea la apelul uneia din metodele statice ale clasei: getByName(), getLocalHost() sau getAllByName() care retunează referința la obiectul creat. De exemplu, prototipul metodei getByName() este următorul:

```
public InetAddress getByName (String host) throws UnknownHostException;
```

În general, o conexiunea de rețea se realizează dinspre o stație (host) numită client, către o altă stație, numită server. Pentru ca un client să se poată conecta la un server, are nevoie de două informații: adresa serverului și portul pe care ascultă aplicația la care se dorește conectarea. Clasa care realizează acest lucru în Java se numește java.net.Sockets. Orice obiect de tip Socket, după instanțiere, are asociate două stream-uri:

- un InputStream de pe care se pot citi date venite de la celălalt capăt al conexiunii, accesibil cu metoda Socket.getInputStream();
- un OutputStream pe care se pot scrie date pentru a fi trimise la celălalt capăt al conexiunii, accesibil cu metoda Socket.getOutputStream().

Clasa Socket

Constructori:

- Socket(InetAddress address, int port) - Creaza un soclu si care se conecteaza la adresa IP specificat prin obiectul InetAddress si portul specificat prin parametrul port.
- Socket(InetAddress address, int port, InetAddress localAddr, int localPort) - Crează un soclu si care se conecteaza la adresa IP specificat prin obiectul InetAddress si portul specificat prin parametrul port. Ultimii doi parametri reprezinta adresa clientului si portul pe care acesta comunica .
- Socket(String host, int port) - Creaza un soclu si care se conecteaza la calculatorul host pe portul specificat prin parametrul port.
- Socket(String host, int port, InetAddress localAddr, int localPort) - Creaza un soclu si care se conecteaza la calculatorul host pe portul specificat prin parametrul port. Ultimii doi parametri reprezinta adresa clientului si portul pe care acesta comunica

Metode principale:

- void close() - Inchide soclul.
- InetAddress getInetAddress() - Returneaza adresa la care soclul este conectat
- InputStream getInputStream() - Returneaza un stream de intrare pentru soclu.
- InetAddress getLocalAddress() - Returneaza adresa pe care soclul este creat.
- int getLocalPort() - Returneaza numarul portului local.
- OutputStream getOutputStream() - Returneza un stream de iesire pentru soclu.
- int getPort() - Returneaza portul la care soclul este conectat.

Socket de server

Rolul unui socket de server este de a accepta conexiuni pe la unul sau mai multi clienți. Clasa care implementează acest comportament este java.net.ServerSocket. Pentru ca un ServerSocket să poată funcționa, are nevoie de o singură informație, și anume portul pe care să accepte conexiuni. Pentru a efectua o conectare concretă, clasa conține o metodă numită accept().

Metoda ServerSocket accept() este blocantă! Asta implica faptul că odată intrat în metodă, programul se blochează pînă în momentul în care un client încercă să se conecteze. În acel moment abia metoda accept() întoarce un obiect de tip Socket asociat acelei conexiuni. În acelaș timp, un client nu se poate conecta atît timp cît serverul nu este blocat în metoda accept().

Clasa ServerSocket

- ServerSocket(int port) - Creaza un soclu server pe portul specificat.
- ServerSocket(int port, int backlog) - Creaza un soclu server pe portul specificat. Al doilea parametru indica lungimea cozii de asteptare.

Metode principale:

- Socket accept() - Asculta conexiunile si le accepta.
- void close() - Inchide soclul.
- InetAddress getInetAddress() - Returneaza adresa locala al soclului server.
- int getLocalPort() - Returneaza portul lacare serverul asteapta conexiunile.

3 Formatul de date XML și JSON

Formatul **JSON** (JavaScript Object Notation) este un format de reprezentare a datelor, larg folosit în prezent pentru transmiterea datelor între programe. Datorită modului în care este structurat, limbajul vine cu o dimensiune mai mică a datelor (față de XML) și o viteză de parsare mai mare. Combinând listele și dicționarele (care conțin intrări de forma cheie: valoare), structurile JSON pot fi extinse oricât de mult, menținând flexibilitatea accesării unui anumit atribut al unui obiect.

Pentru serializarea obiectelor in format se utilizeaza biblioteca gson de la google care permite de a converti un obiect din java in reprezentarea sa in JSON, si la fel permite de a converti un sir de caractere in obiect java. Ea poate lucra cu obiecte arbitrare java, chiar si daca nu avem la dospozitie declararia clasei, Acesta este un un librarie open-source, Cel mai des se utilizeaza anotatii in clasele noastre pentru a notificare care cimpuri sa fie serializate. La procesul de serializare se utilizeaza procesul de reflexie cu ajutorul caruia se citesc metadatele despre cimpurile si metodele clasei, daca nu am mentionat nici o anotatie, ca in cazul nostru atunci vor fi serializare toate cimpurile clasei.

Scopul librariei Gson este:

- propune o implementare simplă, toJson() si fromJson care convertesc în Json și înapoi;
- deasemenea si obiecte existente nemodificabile pot fi convertite în și din Json;
- suportă genericile în java;
- reprezentare simpla a obiectelor.

Pentru a putea utiliza in proiectul nostru libraria Gson am adaugat in maven dependenta care va fi adaugata la proiect:

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.4</version>
</dependency>
```

Documentele XML sunt realizate din unitati de stocare numite entitati, ce contin date parsate sau neparsate. Datele parsate sunt realizate din caractere, unele dintre ele formand date caracter iar altele ca marcaje. Marcajele codifica o descriere a schemei de stocare a documentului si structura logica. XML furnizeaza un mecanism pentru a impune constringeri asupra schemei de stocare si a structurii logice.

XML a fost elaborat pentru:

- separarea **sintaxei** de **semantica** pentru a furniza un cadru comun de structurare a informatiei;
- **construirea de limbaje de mark-up** pentru aplicatii din orice domeniu;
- **structurarea informatiei** in viitor;
- **asigurarea independentiei** de platforma si suport pentru **internationalizare.**

Un document XML este un **arbore ordonat etichetat**:

- **date caracter** - noduri frunza ce contin datele;
- noduri **elemente** etichetate cu;
- un nume (adesea numit si tipul elementului) si;
- multime de atribute, fiecare din ele avand un nume si o valoare.

Acestea pot contine unu sau mai multi copii.

Pentru a serializa obiectele in xml si inapoi, am utilizat libraria JAXB (Java Architecture for XML Binding) oferă o modalitate de conversie între aplicaţii dezvoltate în limbajul de programare Java şi scheme XML. JAXB este o ideologie mai noua in comparatie cu JAXP, aici ideia este de a lega un fisier xml si un obiect java.

JAXB pune la dispoziţie un utilitar pentru compilarea unei scheme XML într-o clasă Java – denumit xjc –, un utilitar de generare a unei scheme XML dintr-o clasă Java – denumit schemagen şi un cadru general de rulare. Astfel, se poate porni de la o definiţie a unei scheme XML (XSD – XML Schema Definition), creându-se obiecte JavaBeans corespunzătoare (folosind compilatorul pentru scheme XML xjc) sau se poate porni de la obiecte JavaBeans creându-se definiţiile unei scheme XML corespunzătoare (folosind utilitarul schemagen). Atât xjc cât şi schemagen fac parte din distribuţia Java, începând cu versiunea 6.

După stabilirea corespondenţei dintre schema XML şi clasele Java, conversia între ele este realizată în mod automat prin intermediul JAXB. Informaţiile reţinute în cadrul documentelor XML pot fi accesate fără a fi necesară înţelegerea structurii lor. Clasele Java rezultate pot fi folosite pentru accesarea serviciului web dezvoltat. Adnotările conţin toate datele necesare pentru ca JAXB să realizeze conversia dintr-un format într-altul, astfel încât obiectele să poată fi transmise prin infrastructura de comunicaţie. JAXB asigură împachetarea obiectelor Java spre documente XML şi despachetarea documentelor XML în instanţe ale claselor Java. Totodată, JAXB poate oferi validarea documentelor XML conform schemei generate, astfel încât acestea respectă constrângerile care au fost stabilite.

4 Realizarea

Repositoriul pe github: https://github.com/artvovc/lab_2_PAD. Proietul este realizat în maven utilizînd java 1.8 și kotlin 1.0.3. Agentul de mesaje reprezintă un server legat de un end-point care ascultă asincron transmițătorii (sender).

```
AsynchronousServerSocketChannel server = AsynchronousServerSocketChannel.open(null);
server.bind(new InetSocketAddress("localhost",5051));
server.accept(attachment,new ConnectionHandler());
```

Codul de sus formează un socket server care se suspendă pe portul 5051 cu adresa locala și acceptă transmițătorii de care se va ocupa obiectul ConnectionHandler. În ConnectionHandler se aceptă următorul transmițător și paralel se formează un canal prin care comunică serverul cu transmițătorul deja conectat. Vezi figura 4.1 în care este prezentat un demo de utilizare a sistemului.

```
MAVEN {"nodeId":"nodeThree","whoRequest":null,"requestType":null,"fromPort":8888,"toPort":1502,"fromHostname":"1
{"nodeId":"nodeThree","whoRequest":null,"requestType":null,"fromPort":8888,"toPort":1502,"fromHostname":"127.0.0
{"nodeId":"nodeFour","whoRequest":null,"requestType":null,"fromPort":8888,"toPort":1502,"fromHostname":"127.0.0.
{"nodeId":"nodeFive","whoRequest":null,"requestType":null,"fromPort":8888,"toPort":1502,"fromHostname":"127.0.0.
{"nodeId":"nodeTwo","whoRequest":null,"requestType":null,"fromPort":8888,"toPort":1502,"fromHostname":"127.0.0.1
Sorted by ID
Empl ( id=0, firstname=jora, lastname=joranovic, age=21, salary=20000, createdDate=1477632744791);
Empl ( id=1, firstname=wartanov, lastname=next, age=20, salary=211, createdDate=1477632744791);
Empl ( id=4, firstname=art, lastname=vb, age=19, salary=444444, createdDate=1477632744779);
Empl ( id=5, firstname=tra, lastname=bv, age=14, salary=21555541, createdDate=1477632744779);
Empl ( id=6, firstname=arte, lastname=evb, age=19, salary=44442244, createdDate=1477632744776);
Empl ( id=7, firstname=trae, lastname=bev, age=34, salary=45541, createdDate=1477632744776);
Empl ( id=8, firstname=trwerta, lastname=bertyev, age=14, salary=21545551, createdDate=1477632744774);
Empl ( id=9, firstname=awerrt, lastname=vyrtyb, age=19, salary=441234444, createdDate=1477632744774);
Empl ( id=10, firstname=tuertuerra, lastname=bvuetruertu, age=14, salary=215255541, createdDate=1477632744771);
```

Figura 4.1 – Demo

Primul rînd este prezentat nodul-maven care știe despre mai multe noduri. Următoarele rînduri la număr patru sunt informațiile ce a transmis mavenul spre client. Restul mai jos este procesarea datelor primite.

```
Node nodeClient = new Node(8888,1502,"127.0.0.1","233.0.0.1");
nodeClient.set_TCPServerPort(5051);
nodeClient.setNodeId("nodeClient");
nodeClient.setWhoRequest(WhoRequest.USER);
nodeClient.setMessage("maven");
nodeClient.setOnlyListen(true);
nodeClient.setEmpls(Arrays.asList(
        new Empl(0,"jora","joranovic",21,20000, new Date().getTime()),
        new Empl(1,"wartanov","next",20,211,new Date().getTime())
));
Thread client = new Thread(nodeClient::runListeningToUDPServer);
client.start();
nodeClient.request();
```

Codul de sus prezintă starea nodului client, acesta are nume unic nodeClient și date locale despre unii lucrători. Nodul client face un request pe ip adressa 233.0.0.1 care este o ip adresă broadcast pentru ca toate nodurile să primească acest mesaj.

Fiecare nod primește mesajul maven, după care se transmit clientului toată informația despre nodul cutare. Clientul decide cinei maven și înceracă să realizeze conexiune sigură cu nodul maven. După ce sa format un handshake se cere la nodul-maven toate datele sale și a vecinilor, ăsta la rîndul lui cere datele de la vecini, le grupează și le transmite clientului prin canal sigur.

```
System.out.println("Sorted by ID");
Database.getInstance().getEmplList().stream().sorted((e1, e2) ->
Integer.compare(e1.getId(), e2.getId())).forEach(System.out::println);
System.out.println("Filtered by Age < 30");
Database.getInstance().getEmplList().stream().filter(empl ->
empl.getAge()<30).forEach(System.out::println);
System.out.println("Grouped by Age");
Map<Integer, List<Empl>> totalByDept
        = Database.getInstance().getEmplList().stream()
        .collect(Collectors.groupingBy(Empl::getAge));
System.out.println(totalByDept);
```

Stream api ușurează procesarea datelor prin minimizarea codului și scalabilitatea de acces a serviciilor strimurilor. În java la stream api este unicul neajuns că este posibil de procesat doar colecții pe cînd la linq în c# se pot și procesa xml și json fișiere.

Procesarea datelor în formatul xml în java. Java suportă dom, sax și stax parsări, dar mai oferă și JAXB o tehnologie ușoară de parsare a xml datelor. În laborator am utilizat DOM parser al javei care se află în libraria DocumentBuilderFactory, SAX în SAXParserFactory și JAXB care se află în libraria jaxb-api și jaxp-impl.

DOM parsarea are posibilități de a înscrie și de a parsa date xml, pe cînd SAX poate doar să citească (parseze) deoarece e bazat pe evenimente. JAXB este un mecanism puternit ce utilizează marshalizări și în componență cu anotații poate forma date xml cît și invers formatarea datelor xml în obiecte.

```
package com.model;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * Created by Artemie on 09.10.2016.
 */
@XmlRootElement(name = "empl")
@XmlAccessorType(XmlAccessType.FIELD)
public class Empl {
    private Integer id;
    private String firstname;
```

```
    private String lastname;
    private Integer age;
    private Integer salary;
    private Long createdDate;

}
```

Acestea sunt cîmpurile datelor care vor fi stocate în fișiere sau prezentate pe consolă.

Mai jos în figura 4.2 este reprezentat rezultatul unei din parsări.

```
[INFO] -> DOM OUTPUT
[INFO] -> DOM print from created document on console


<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<empls>
    <empl>
        <id>0</id>
        <firstname>jora</firstname>
        <lastname>joranovic</lastname>
        <age>21</age>
        <salary>20000</salary>
        <createdDate>1477632744791</createdDate>
    </empl>
    <empl>
        <id>1</id>
        <firstname>wartanov</firstname>
        <lastname>next</lastname>
        <age>20</age>
        <salary>211</salary>
        <createdDate>1477632744791</createdDate>
    </empl>
    <empl>
        <id>9</id>
        <firstname>awerrt</firstname>
        <lastname>vyrtyb</lastname>
        <age>19</age>
        <salary>441234444</salary>
        <createdDate>1477632744774</createdDate>
    </empl>
    <empl>
        <id>8</id>
```

Figura 4.2 – Rezultatul parsării

Pentru detalii accesați anexele, github repozitoriul și bibliografia.

Concluzia

Lucrarea dată a avut ca scop să ne facă cunoscuți cu problemel ce pot apărea la transmiterea datelor prin canale distribuite. Orice nod din rețea dacă să-l nosiderăm ca o filială unei companii conține o structură de informații similară, deci sunt și acolo aceeași lucrători dar cu diferite salarii, nume, prenume și restul. La ce conduc că dacă avem mai multe filiale unei companii și dorim ca să strîngem toată informația de pe toate nodurile întrun nod anumit. Se întîlnește o problemă de conectare, unele noduri de exemplu sunt tare departe pentru ca să poată fi unite fizic. Validarea datelor tot este o problemă, cît și procesarea lor.

DOM parser are un neajuns simțitor din mutiv că stocheza datele în memorie timpurie sub formă de arbore și știm că parcurgerea unui arbore foarte mare va necesita mult timp. SAX în comparație cu DOM este mai rapid deoarece utilizează evenimentele în felul următor: cînd se citește fișierul xml se analizează denumirea tăgurilor și dacă denumirea această este procesată în starteveniment atunci e posibil de primit și valoare acestui tăg. JAXB sau Java Architecture for XML Binding este cel mai des utilizat de javiști din motiv că este foarte simplu și productiv utilizarea acestei librării deoarece oferă unelte actuale ca annotațiile și doar două funcții importante de marshalizare și demarshalizare anterior trebuie de indicat tipul clasei și asta este totul.

Bibliografia

1 **Tanenbaum, Andrew S. and Van Steen, Maarten.** *Distributed systems: principles and paradigms.* s.l. : Pearson Prentice Hall, 2007.

2 **Bruce Eckel**, Thinking in Java (4rd Edition). http://www.mindview.net/Books/ Concurrency

3 **Brian Goetz, Tim Peierls** , Java Concurrency In Practice March 2006

4 **Sun Microsystem Inc**., Online Java Tutorials,
https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html
http://docs.oracle.com/javase/tutorial/networking/sockets/

5 **Kenneth L. Calvert and Michael J. Donahoo**. TCP/IP Sockets in Java™: Practical Guide for Programmers

6 XML java. [regim de acces]: http://www.mkyong.com/tutorials/java-xml-tutorials/

Anexe A
Node

```
package com.node;

import com.database.Database;
import com.enums.RequestType;
import com.enums.WhoRequest;
import com.model.Empl;
import com.model.Model;
import com.util.JSONUtil;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.*;
import java.nio.channels.AsynchronousServerSocketChannel;
import java.util.List;
import java.util.Objects;

/**
 * Created by Artemie on 04.10.2016.
 */
public class Node {
    private String nodeId;
    private Boolean onlyListen = Boolean.FALSE;
    private int _UDPServer_port;
    private int _MulticastSocket_port;//5051
    private String hostname;
    private String _MulticastSocket_broadcastIp;//233.0.0.1
    private DatagramSocket serverSocket;
    private SocketAddress socketAddress;
    private MulticastSocket multicastSocket;
    private WhoRequest whoRequest;
    private RequestType requestType;
    private Model model = new Model();
    private String message;
    private int countConnections;
    private Model maven = new Model();
    private int countNodes;
    private List<String> knownNodes;
    private int _TCPServerPort;


    public Node(String hostname, int _UDPServer_port) throws Exception {
        this._UDPServer_port = _UDPServer_port;
        this.hostname = hostname;
    }

    public Node(int _UDPServer_port, int _MulticastSocket_port, String hostname, String
_MulticastSocket_broadcastIp) {
        this._UDPServer_port = _UDPServer_port;
        this._MulticastSocket_port = _MulticastSocket_port;
        this.hostname = hostname;
        this._MulticastSocket_broadcastIp = _MulticastSocket_broadcastIp;
    }

    private void UDPServerStandardConfig() throws Exception {
        socketAddress = new InetSocketAddress(this.hostname, this._UDPServer_port);
```

```
        serverSocket = new DatagramSocket(this._UDPServer_port);
        serverSocket.setBroadcast(true);
        maven.setCountConnections(0);
    }

    public void runUDPServer() {
        try {
            UDPServerStandardConfig();
            int temp = 0;
            while (true) {
                byte[] receiveBuff = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(receiveBuff,
receiveBuff.length);
                serverSocket.receive(receivePacket);
                String msg = new String(receivePacket.getData());
                Model model = (Model) JSONUtil.getJAVAObjectfromJSONString(msg,
Model.class);

                if (model.getWhoRequest() == WhoRequest.USER &&
Objects.equals(model.getMessage(), "maven")) {
                    model = new Model();
                    model.setWhoRequest(WhoRequest.NODE);
                    model.setMessage("getCount");
                    msg = JSONUtil.getJSONStringfromJAVAObject(model);
                    DatagramPacket sendPacket = new DatagramPacket(msg.getBytes(),
msg.getBytes().length, InetAddress.getByName("233.0.0.1"), receivePacket.getPort());
                    serverSocket.send(sendPacket);
                }
                if (Objects.equals(model.getMessage(), "count")) {
                    if (maven.getCountConnections() < model.getCountConnections())
                        maven = model;
                    temp++;
                    msg = JSONUtil.getJSONStringfromJAVAObject(new Model());
                    if (temp == countNodes) {
                        maven.setMessage("Imaven");
                        msg = JSONUtil.getJSONStringfromJAVAObject(maven);
                        DatagramPacket sendPacket = new DatagramPacket(msg.getBytes(),
msg.getBytes().length, InetAddress.getByName("233.0.0.1"), receivePacket.getPort());
                        serverSocket.send(sendPacket);
                    }
                }

                if (model.getWhoRequest() == WhoRequest.USER &&
Objects.equals(model.getMessage(), "getTuples")) {
                    this.maven.setMessage("sendTuplesToClient");
                    this.maven.setTcpServerPort(model.getTcpServerPort());
                    msg = JSONUtil.getJSONStringfromJAVAObject(maven);
                    DatagramPacket sendPacket = new DatagramPacket(msg.getBytes(),
msg.getBytes().length, InetAddress.getByName("233.0.0.1"), receivePacket.getPort());
                    serverSocket.send(sendPacket);
                }

                if (model.getMessage() == null) msg =
JSONUtil.getJSONStringfromJAVAObject(new Model());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    public void runListeningToUDPServer() {
        try {
            multicastSocketConfig();
            byte[] data = null;
            int i = 0;
            while (true) {
                data = new byte[0];
                data = new byte[1024];
                if (!this.onlyListen) {
                    request();
                }
                DatagramPacket receiveDatagramPacket = new DatagramPacket(data,
data.length);
                multicastSocket.receive(receiveDatagramPacket);
                String msg = new String(receiveDatagramPacket.getData());
                Model model = (Model) JSONUtil.getJAVAObjectfromJSONString(msg,
Model.class);
                if (model.getWhoRequest() == WhoRequest.NODE &&
Objects.equals(model.getMessage(), "getCount")) {
                    setModelPack();
                    String msgJson = null;
                    this.model.setMessage("count");
                    msgJson = JSONUtil.getJSONStringfromJAVAObject(this.model);
                    DatagramPacket datagramPacket = new
DatagramPacket(msgJson.getBytes(), msgJson.getBytes().length, new
InetSocketAddress(this.hostname, this._UDPServer_port));
                    multicastSocket.send(datagramPacket);
                }
                if (Objects.equals(this.nodeId, "nodeClient") &&
Objects.equals(model.getMessage(), "Imaven")) {
                    this.maven = model;
                    //AICI STARTEZ TCP SERVER PE UN PORT

                    runTCPServer();

                    this.model.setMessage("getTuples");
                    String msgJson = JSONUtil.getJSONStringfromJAVAObject(this.model);
                    DatagramPacket datagramPacket = new
DatagramPacket(msgJson.getBytes(), msgJson.getBytes().length, new
InetSocketAddress(this.hostname, this._UDPServer_port));
                    multicastSocket.send(datagramPacket);
                    System.out.println("MAVEN " +
JSONUtil.getJSONStringfromJAVAObject(maven));
                }
                if (Objects.equals(model.getMessage(), "sendTuplesToClient")) {
                    if (Objects.equals(this.nodeId, model.getNodeId()) ||
check(this.nodeId, model.getKnownNodes())) {
                        Socket subscriberSocketClient = new
Socket(model.getFromHostname(), model.getTcpServerPort());
                        OutputStream out = subscriberSocketClient.getOutputStream();
                        DataOutputStream DataSend = new DataOutputStream(out);
                        String str = JSONUtil.getJSONStringfromJAVAObject(this.model);
                        byte[] bytes = str.getBytes();
                        DataSend.write(bytes, 0, bytes.length);
                        out.close();
                        DataSend.close();
                        subscriberSocketClient.close();
                    }
```

```
                }
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void runTCPServer() {
        Database.getInstance().setEmplList(this.model.getEmplList());
        new Thread(() -> {
            AsynchronousServerSocketChannel server = null;
            try {
                server = AsynchronousServerSocketChannel.open(null);
                server.bind(new InetSocketAddress("127.0.0.1", this._TCPServerPort));
                Attachment attachment = new Attachment();
                attachment.setServer(server);
                server.accept(attachment, new ConnectionHandler());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();
    }

    private boolean check(String nodeId, List<String> knownNodes) {
        for (String knownNode : knownNodes)
            if (Objects.equals(knownNode, nodeId)) return true;
        return false;
    }

    private void multicastSocketConfig() {
        this.multicastSocket = null;
        try {
            this.multicastSocket = new MulticastSocket(this._MulticastSocket_port);
            this.multicastSocket.setBroadcast(true);

this.multicastSocket.joinGroup(InetAddress.getByName(this._MulticastSocket_broadcastIp)
);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void setModelPack() {
        this.model.setNodeId(this.nodeId);
        this.model.setMessage(this.message);
        this.model.setWhoRequest(this.whoRequest);
        this.model.setFromHostname(this.hostname);//localhost
        this.model.setFromPort(this._UDPServer_port);//8888
        this.model.setToHostname(this._MulticastSocket_broadcastIp);//233.0.0.1
        this.model.setToPort(this._MulticastSocket_port);//1502
        this.model.setKnownNodes(this.knownNodes);
        this.model.setCountConnections(this.countConnections);
        this.model.setTcpServerPort(this._TCPServerPort);
    }

    public void request() {
        setModelPack();
        String msgJson = null;
```

```
        try {
            msgJson = JSONUtil.getJSONStringfromJAVAObject(model);
            DatagramPacket datagramPacket = new DatagramPacket(msgJson.getBytes(),
msgJson.getBytes().length, new InetSocketAddress(this.hostname, this._UDPServer_port));
            multicastSocket.send(datagramPacket);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public String getNodeId() {
        return nodeId;
    }

    public void setNodeId(String nodeId) {
        this.nodeId = nodeId;
    }

    public void setOnlyListen(Boolean onlyListen) {
        this.onlyListen = onlyListen;
    }

    public void setEmpls(List<Empl> empls) {
        this.model.setEmplList(empls);
    }

    public WhoRequest getWhoRequest() {
        return whoRequest;
    }

    public void setWhoRequest(WhoRequest whoRequest) {
        this.whoRequest = whoRequest;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public int getCountConnections() {
        return countConnections;
    }

    public void setCountConnections(int countConnections) {
        this.countConnections = countConnections;
    }

    public int getCountNodes() {
        return countNodes;
    }

    public void setCountNodes(int countNodes) {
        this.countNodes = countNodes;
    }

    public List<String> getKnownNodes() {
```

```
        return knownNodes;
    }

    public void setKnownNodes(List<String> knownNodes) {
        this.knownNodes = knownNodes;
    }

    public int get_TCPServerPort() {
        return _TCPServerPort;
    }

    public void set_TCPServerPort(int _TCPServerPort) {
        this._TCPServerPort = _TCPServerPort;
    }
}
```

Anexa B

Attachment, ConnectionHandler, ReadHandler

```
package com.node;

import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousServerSocketChannel;
import java.nio.channels.AsynchronousSocketChannel;

/**
 * Created by Artemie on 03.09.2016.
 */
public class Attachment {
    private AsynchronousServerSocketChannel server;
    private AsynchronousSocketChannel client;
    private ByteBuffer buffer;

    public AsynchronousServerSocketChannel getServer() {
        return server;
    }

    public void setServer(AsynchronousServerSocketChannel server) {
        this.server = server;
    }

    public AsynchronousSocketChannel getClient() {
        return client;
    }

    public void setClient(AsynchronousSocketChannel client) {
        this.client = client;
    }

    public ByteBuffer getBuffer() {
        return buffer;
    }

    public void setBuffer(ByteBuffer buffer) {
        this.buffer = buffer;
    }
}


package com.node;

import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;

/**
 * Created by Artemie on 03.09.2016.
 */
public class ConnectionHandler implements CompletionHandler<AsynchronousSocketChannel,
Attachment> {



    @Override
    public void completed(AsynchronousSocketChannel client, Attachment attachment) {
```

```
        attachment.getServer().accept(attachment,this);
        Attachment newAttachment = new Attachment();
        newAttachment.setServer(attachment.getServer());
        newAttachment.setClient(client);
        newAttachment.setBuffer(ByteBuffer.allocate(2048));
        client.read(newAttachment.getBuffer(), newAttachment,new ReadHandler());
    }

    @Override
    public void failed(Throwable exc, Attachment attachment) {

    }
}


package com.node;

import com.database.Database;
import com.model.Model;
import com.util.JSONUtil;

import java.nio.channels.CompletionHandler;
import java.nio.charset.Charset;

class ReadHandler implements CompletionHandler<Integer, Attachment> {
    @Override
    public void completed(Integer result, Attachment attachment) {
        attachment.getBuffer().flip();
        int limits = attachment.getBuffer().limit();
        byte bytes[] = new byte[limits];
        attachment.getBuffer().get(bytes, 0, limits);
        attachment.getBuffer().rewind();
        Charset cs = Charset.forName("UTF-8");
        String msg = new String(bytes, cs);
        Model model = null;
        try {
            model = (Model) JSONUtil.getJAVAObjectfromJSONString(msg,Model.class);
            System.out.println(msg);
            Database.getInstance().getEmplList().addAll(model.getEmplList());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void failed(Throwable e, Attachment attachment) {
        e.printStackTrace();
    }
}
```

Anexa C
Model, Empl, Database

```
package com.model;

/**
 * Created by Artemie on 09.10.2016.
 */
public class Empl {
    private Integer id;
    private String firstname;
    private String lastname;
    private int age;
    private int salary;
    private Long createdDate;

    public Empl(){}

    public Empl(Integer id, String firstname, String lastname, int age, int salary,
Long createdDate) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
        this.age = age;
        this.salary = salary;
        this.createdDate = createdDate;
    }

    @Override
    public String toString() {
        return "Empl ( id="+id+", firstname="+firstname+", lastname"+lastname+",
age="+age+", salary="+salary+", createdDate="+createdDate+");";
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public int getAge() {
```

```
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }

    public Long getCreatedDate() {
        return createdDate;
    }

    public void setCreatedDate(Long createdDate) {
        this.createdDate = createdDate;
    }
}
```

```
package com.model;

import com.enums.RequestType;
import com.enums.WhoRequest;

import java.util.List;

/**
 * Created by Artemie on 04.10.2016.
 */
public class Model {
    private String nodeId;
    private WhoRequest whoRequest;
    private RequestType requestType;
    private int fromPort;
    private int toPort;//5051
    private String fromHostname;
    private String toHostname;//233.0.0.1
    private String message;
    private List<Empl> emplList;
    private int countConnections;
    private List<String> knownNodes;
    private int tcpServerPort;

    public String getNodeId() {
        return nodeId;
    }

    public void setNodeId(String nodeId) {
        this.nodeId = nodeId;
    }

    public WhoRequest getWhoRequest() {
        return whoRequest;
```

```
}

public void setWhoRequest(WhoRequest whoRequest) {
    this.whoRequest = whoRequest;
}

public RequestType getRequestType() {
    return requestType;
}

public void setRequestType(RequestType requestType) {
    this.requestType = requestType;
}

public String getFromHostname() {
    return fromHostname;
}

public void setFromHostname(String fromHostname) {
    this.fromHostname = fromHostname;
}

public List<Empl> getEmplList() {
    return emplList;
}

public void setEmplList(List<Empl> emplList) {
    this.emplList = emplList;
}

public int getFromPort() {
    return fromPort;
}

public void setFromPort(int fromPort) {
    this.fromPort = fromPort;
}

public int getToPort() {
    return toPort;
}

public void setToPort(int toPort) {
    this.toPort = toPort;
}

public String getToHostname() {
    return toHostname;
}

public void setToHostname(String toHostname) {
    this.toHostname = toHostname;
}

public String getMessage() {
    return message;
}

public void setMessage(String message) {
```

```
        this.message = message;
    }

    public int getCountConnections() {
        return countConnections;
    }

    public void setCountConnections(int countConnections) {
        this.countConnections = countConnections;
    }

    public List<String> getKnownNodes() {
        return knownNodes;
    }

    public void setKnownNodes(List<String> knownNodes) {
        this.knownNodes = knownNodes;
    }

    public int getTcpServerPort() {
        return tcpServerPort;
    }

    public void setTcpServerPort(int tcpServerPort) {
        this.tcpServerPort = tcpServerPort;
    }
}
```

```
package com.database;

import com.model.Empl;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Created by Artemie on 04.10.2016.
 */
public class Database {
    private static Database ourInstance = new Database();

    public static Database getInstance() {
        return ourInstance;
    }

    private List<Empl> emplList = Collections.synchronizedList(new ArrayList<Empl>());

    private Database() {
    }

    public List<Empl> getEmplList() {
        return emplList;
    }
    public void setEmplList(List<Empl> emplList) {
        this.emplList = Collections.synchronizedList(new ArrayList<Empl>(emplList));
    }
}
```

Anexa D

XMLFormater, SAXParser, DOMParser, JAXBParser

```
package com.util.xml;

import com.model.Empl;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by Artemie on 14.10.2016.
 */

public final class XMLDOMUtil {
    DocumentBuilderFactory docFactory;
    DocumentBuilder documentBuilder;
    Document doc;
    List<Empl> empls;

    public XMLDOMUtil(List<Empl> empls) throws ParserConfigurationException {
        docFactory = DocumentBuilderFactory.newInstance();
        documentBuilder = docFactory.newDocumentBuilder();
        doc = documentBuilder.newDocument();
        this.empls = empls;
    }

    private void appendSimpleNode(Element root, String nameNode, Object value) {
        Element element = doc.createElement(nameNode);
        element.appendChild(doc.createTextNode((String)value));
        root.appendChild(element);
    }

    private void appendComplexListNode(Element root, String nameNode, List<Empl>
values) {
        values.forEach(empl -> {
            Element listElement = doc.createElement(nameNode);
            appendSimpleNode(listElement,"id",empl.getId().toString());
            appendSimpleNode(listElement,"firstname",empl.getFirstname());
            appendSimpleNode(listElement,"lastname",empl.getLastname());
            appendSimpleNode(listElement,"age",empl.getAge().toString());
            appendSimpleNode(listElement,"salary",empl.getSalary().toString());

appendSimpleNode(listElement,"createdDate",empl.getCreatedDate().toString());
            root.appendChild(listElement);
        });

    }

    public Document createXMLDoc() {
        Element root = doc.createElement("empls");
        doc.appendChild(root);
```

```
        appendComplexListNode(root,"empl",this.empls);
        return doc;
    }

    public List<Empl> createListFromDoc(Document doc){
        this.empls = new ArrayList<Empl>();

//        Element elem = doc.getElementById("empls");

        NodeList nodes = doc.getElementsByTagName("empl");

        for (int i = 0; i < nodes.getLength(); i++) {
            Node tempNode = nodes.item(i);
            Element tempElement = (Element) tempNode;
            Empl tempEmpl = new Empl();


tempEmpl.setId(Integer.parseInt(tempElement.getElementsByTagName("id").item(0).getTextC
ontent()));

tempEmpl.setFirstname(tempElement.getElementsByTagName("firstname").item(0).getTextCont
ent());

tempEmpl.setLastname(tempElement.getElementsByTagName("lastname").item(0).getTextConten
t());

tempEmpl.setAge(Integer.parseInt(tempElement.getElementsByTagName("age").item(0).getTex
tContent()));

tempEmpl.setSalary(Integer.parseInt(tempElement.getElementsByTagName("salary").item(0).
getTextContent()));

tempEmpl.setCreatedDate(Long.parseLong(tempElement.getElementsByTagName("createdDate").
item(0).getTextContent()));

            this.empls.add(tempEmpl);
        }
        return this.empls;
    }

}

package com.util.xml;

import org.w3c.dom.Document;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.StringReader;
import java.io.StringWriter;

/**
```

```
 * Created by Artemie on 14.10.2016.
 */
public class XMLFormater {

    public static String printXMLString(String input) throws Exception {
        Source xmlInput = new StreamSource(new StringReader(input));
        StringWriter stringWriter = new StringWriter();
        StreamResult xmlOutput = new StreamResult(stringWriter);
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
        transformer.setOutputProperty(OutputKeys.METHOD, "xml");
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
        transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount",
"4");
        transformer.transform(xmlInput, xmlOutput);
        return xmlOutput.getWriter().toString();

    }

    public static void printDocument(Document doc, OutputStream out) throws Exception {
        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer transformer = tf.newTransformer();
        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
        transformer.setOutputProperty(OutputKeys.METHOD, "xml");
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
        transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount",
"4");
        transformer.transform(new DOMSource(doc), new StreamResult(new
OutputStreamWriter(out, "UTF-8")));
    }

}

package com.util.xml;

import com.model.XmlReprezentationEmpls;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import java.io.File;

/**
 * Created by Artemie on 14.10.2016.
 */
public class XMLJAXBUtil {
    private JAXBContext jaxbContext = null;
    private Marshaller jaxbMarshaller = null;
    private Unmarshaller jaxbUnmarshaller = null;
    private XmlReprezentationEmpls empls = null;
    private File file = null;

    public XMLJAXBUtil(XmlReprezentationEmpls empls, String file) throws Exception {
        jaxbContext = JAXBContext.newInstance(XmlReprezentationEmpls.class);
        jaxbMarshaller = jaxbContext.createMarshaller();
        jaxbUnmarshaller = jaxbContext.createUnmarshaller();
```

```
        jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        this.empls = empls;
        this.file = new File(file);
    }

    public void printConsole() throws Exception {
        jaxbMarshaller.marshal(this.empls, System.out);
    }

    public void printToFile() throws Exception {
        jaxbMarshaller.marshal(this.empls, this.file);
    }

    public XmlReprezentationEmpls readFromFile(String file) throws Exception {
        return (XmlReprezentationEmpls) jaxbUnmarshaller.unmarshal(new File(file));
    }
}
```

```
package com.util.xml;

import com.model.XmlReprezentationEmpls;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import java.io.File;

/**
 * Created by Artemie on 14.10.2016.
 */
public class XMLJAXBUtil {
    private JAXBContext jaxbContext = null;
    private Marshaller jaxbMarshaller = null;
    private Unmarshaller jaxbUnmarshaller = null;
    private XmlReprezentationEmpls empls = null;
    private File file = null;

    public XMLJAXBUtil(XmlReprezentationEmpls empls, String file) throws Exception {
        jaxbContext = JAXBContext.newInstance(XmlReprezentationEmpls.class);
        jaxbMarshaller = jaxbContext.createMarshaller();
        jaxbUnmarshaller = jaxbContext.createUnmarshaller();
        jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        this.empls = empls;
        this.file = new File(file);
    }

    public void printConsole() throws Exception {
        jaxbMarshaller.marshal(this.empls, System.out);
    }

    public void printToFile() throws Exception {
        jaxbMarshaller.marshal(this.empls, this.file);
    }

    public XmlReprezentationEmpls readFromFile(String file) throws Exception {
        return (XmlReprezentationEmpls) jaxbUnmarshaller.unmarshal(new File(file));
    }
}
```