ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

Τεύχος Προδιαγραφών Σχεδίασης
JAIN SIP Proxy και SIP Communicator

Έκδοση:	1.0
Ημερομηνία Εκτύπωσης:	21/11/2014
Ημερομηνία Έκδοσης:	28/11/2014
Κατάσταση Έκδοσης:	Τελικό
Κατάσταση Έγκρισης:	Εγκρίθηκε
Εγκρίθηκε από:	Καθ. Κ. Κοντογιάννης
Προετοιμάστηκε από:	Λώλος Κωνσταντίνος Μπουρίκας Φοίβος Ποδηματά Χαρίκλεια
Ελέγχθηκε από:	Καθ. Κ. Κοντογιάννης
Διαδρομή Αρχείου:	
Όνομα Αρχείου:	SDD.pdf
Αριθμός Εγγράφου:	

Περιεχόμενα

1 Εισαγωγή

1.1 Επισκόπηση

Εδώ προσθέσαμε κάποιες επεκτάσεις που αφορούν το πρωτόκολλο SIP και συγκεκριμένα «επεκτείνουν» την εφαρμογή JAIN-SIP-PRESENCE-PROXY και SIP - communicator για το VoIP. Επιγραμματικά, οι επεκτάσεις που έγιναν ήταν οι ακόλουθες:

i **Εγγραφή κάποιου χρήστη στο σύστημα (Register).** Ο χρήστης, προκειμένου να συνδεθεί στο σύστημα, πρέπει να είναι εγγεγραμμένος. Αν είναι εγγεγραμμένος, κάνει login, αλλιώς πρέπει να κάνει register.

ii **Αποκλεισμός κάποιου χρήστη (Block).** Ο χρήστης A μπορεί από τη λίστα των διαθέσιμων χρηστών να διαλέξει έναν άλλον χρήστη B (ή και περισσότερους) και να τους «αποκλείσει», δηλαδή, να μην είναι δυνατό να δεχθεί κλήση από αυτούς. Όταν κάποιος από τους *αποκλεισμένους* χρήστες δοκιμάσει να τον καλέσει, παίρνει ως απάντηση ένα μήνυμα μη-διαθεσιμότητας.

iii **Προώθηση Κλήσης (Forwarding).** Ο χρήστης A μπορεί να ορίσει έναν χρήστη στον οποίον να προωθούνται οι κλήσεις του. Δεν είναι δυνατό ένας χρήστης να προωθεί σε δύο άλλους χρήστες ταυτόχρονα. Επιπλέον, σε περιπτώσεις κυκλικής προώθησης, η προώθηση καταλήγει στον τελευταίο χρήστη των προωθήσεων.

iv **Χρέωση Κλήσης (Billing).** Ο καλών χρήστης χρεώνεται το ποσό της κλήσης και η χρέωση είναι είτε βαθμωτή με βάση τη διάρκεια της χρέωσης, είτε σταθερή, ή δωρεάν. Ο κάθε χρήστης μπορεί να έχει διαφορετική τιμολογιακή πολιτική, με βάση την οποία να χρεώνεται η κλήση.

Οι επεκτάσεις μας είναι συμβατές με το πρότυπο RFC 3261 και βασίστηκαν στα λειτουργικά χαρακτηριστικά του προτύπου. Προβληματικές καταστάσεις, όπως καταστροφή του προγράμματος, πρόβλημα στον Proxy Server και άλλα, τα διαχειριζόμαστε στα πλαίσια του προτύπου αυτού και τα αναλύουμε εκτενέστερα παρακάτω.

1.2 Αναφορές

- `http://www.iptel.org/`

- http://www.iptel.org/files/sip_tutorial.pdf
- http://www.radvision.com/NR/rdonlyres/51855E82-BD7C-4D9D-AA8A-E822E3F
  0/RADVISIONSIPProtocolOverview.pdf
- http://courses.softlab.ntua.gr/softeng//#lectures

2 Κύριες Σχεδιαστικές Αποφάσεις

Καταρχάς, λόγω των επεκτάσεων που θέλαμε να υλοποιήσουμε χρειαζόμασταν έναν τρόπο να αποθηκεύουμε τόσο τους χρήστες της εφαρμογής, όσο και πληροφορίες σχετικές με αυτούς. Για το λόγο αυτό, καταλήξαμε στην απόφαση να χρησιμοποιηθεί μία *βάση δεδομένων*. Επιλέξαμε, λοιπόν, την MySQL μιας και είναι μία γλώσσα που χρησιμοποιείται αρκετά σήμερα και σε εμπορικές εφαρμογές.

Για την υλοποίηση των επεκτάσεων forwarding, blocking η λογική που ακολουθήσαμε ήταν η εξής: Αρχικά, πρόσβαση στη βάση και διαχείριση αυτής (updates, insertions, deletes) μπορούσε να κάνει *μόνο* ο proxy server. Για το λόγο αυτό, όταν κάποιος χρήστης αιτούνταν κάποια από αυτές τις υπηρεσίες, έστελνε ένα request στον proxy (με κατάλληλο «τύπο», ανάλογα το αίτημα), ο proxy το διαχειριζόταν ανάλογα, ενημέρωνε τη βάση, και κατά περίπτωση, έστελνε «απάντηση» στον communicator.

Billing.

Κατά το billing ένας χρήστης έχει τις εξής δυνατότητες:

i Επιλογή τιμολογιακής πολιτικής

ii Εμφάνιση κόστους κλήσης

Blocking.

Κατά το blocking ένας χρήστης έχει τις εξής δυνατότητες:

i Αποκλεισμός ενός άλλου χρήστη (block).

ii Άρση αποκλεισμού κάποιου χρήστη (unblock).

iii Εμφάνιση της λίστας με τους χρήστες που έχει αποκλείσει.

Όταν ο χρήστης A έχει αποκλείσει τον χρήστη B, τότε όταν ο B προσπαθήσει να καλέσει τον A, θα λάβει απάντηση «μη-διαθέσιμος». Για τις λειτουργίες αυτές, ο communicator στέλνει κατάλληλα requests στον proxy, προκειμένου ο proxy είτε να ελέγξει, είτε να ανανεώσει τη βάση.

Forwarding.

Κατά το forwarding ένας χρήστης έχει τις εξής δυνατότητες:

i Προώθηση των κλήσεών του σε κάποιον άλλον χρήστη. (forward).

ii Άρση της προώθησης των κλήσεων (unforward).

iii Εμφάνιση του χρήστη στον οποίον προωθεί τις κλήσεις του ο χρήστης.

Όταν ο χρήστης A προωθεί στον χρήστη B και τον καλέσει ο χρήστης C, τότε θα χτυπήσει το τηλέφωνο του B απευθείας. Σε περίπτωση «αλυσίδας» προωθήσεων, θα χτυπήσει το τηλέφωνο του τελευταίου χρήστη στην αλυδίδα. Για τις λειτουργίες αυτές, ο communicator στέλνει κατάλληλα requests στον proxy, προκειμένου ο proxy είτε να ελέγξει, είτε να ανανεώσει τη βάση.

Οι νέες λειτουργίες εν γένει υλοποιήθηκαν σε καινούρια πακέτα, αλλά φυσικά, παραλλάξαμε και υπάρχοντα πακέτα για να εξυπηρετήσουμε τις ανάγκες μας για επικοινωνία.

Τέλος, προκειμένου να υποστηριχθούν αυτές οι καινούριες λειτουργίες, επεκτείναμε και το GUI της εφαρμογές, προσθέτοντας τα απαραίτητα παράθυρα, τα οποία παραθέτουμε και σε επόμενη ενότητα.

2.1 Design Patterns

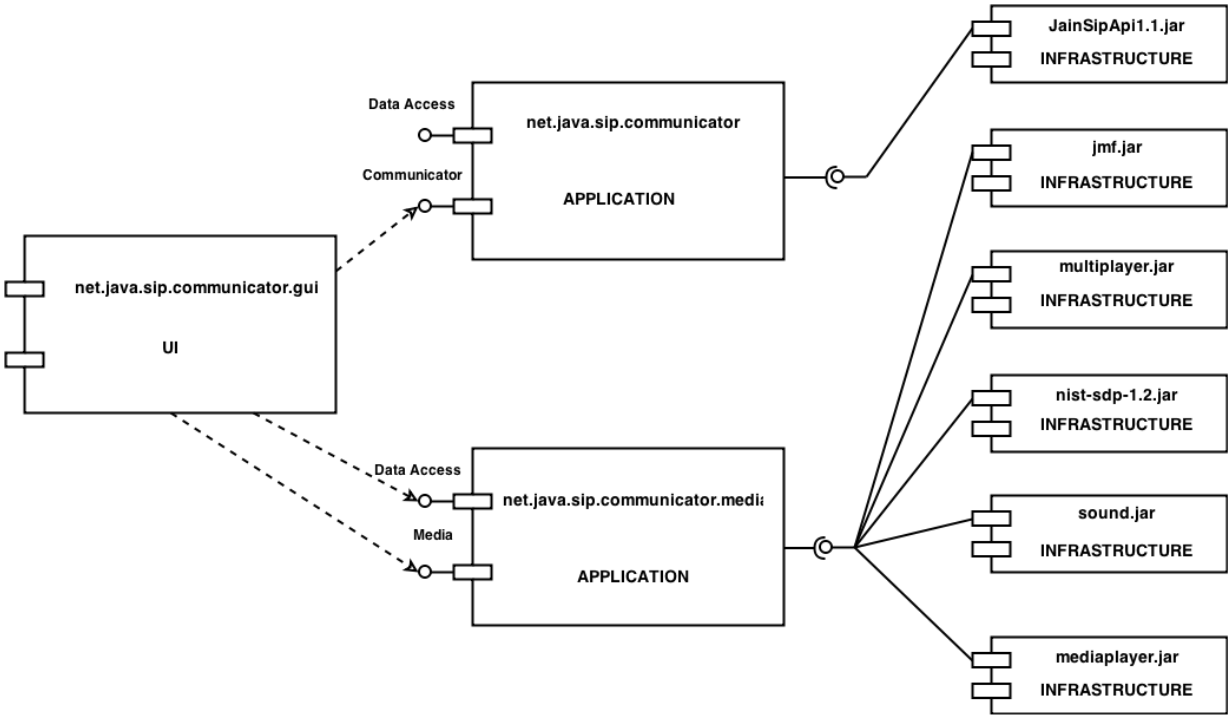
Επιλέξαμε να χρησιμοποιήσουμε κάποια σχεδιαστικά μορφήματα στην υλοποίησή μας. Συγκεκριμένα, για τους Servers χρησιμοποιήθηκε το σχεδιαστικό μόρφωμα singleton γιατί το singleton design pattern περιορίζει το instantiation μιας κλάσης σε ένα μόνο αντικείμενο. Αυτό μας είναι ιδιαίτερα χρήσιμο στους servers, όπου χρειάζεται μόνο ένα αντικείμενο για να καθορίσει τις ενέργειες που θα γίνουν σε ολόκληρο το σύστημα.

Επιπλέον, χρησιμοποιήθηκε το template method design pattern για την αποστολή μηνυμάτων από τον communicator. Το μόρφωμα template method, το οποίο είναι behavioral design pattern, κρίθηκε καταλληλότερο γιατί καθορίζει τον σκελετό του προγράμματος, και μας δίνει τη δυνατότητα στη συνέχεια να αλλάξουμε κάποια βήματα του αλγορίθμου χωρίς, ωστόσο, να αλλάζει η δομή του αλγορίθμου. Αυτό μας φάνηκε ιδιαίτερα χρήσιμο, γιατί στέλναμε πολλά διαφορετικά είδη μηνυμάτων, τα οποία κατασκευάζονταν με τον ίδιο μακροσκελή τρόπο, αλλάζοντας κάθε φορά ελάχιστα βήματα.

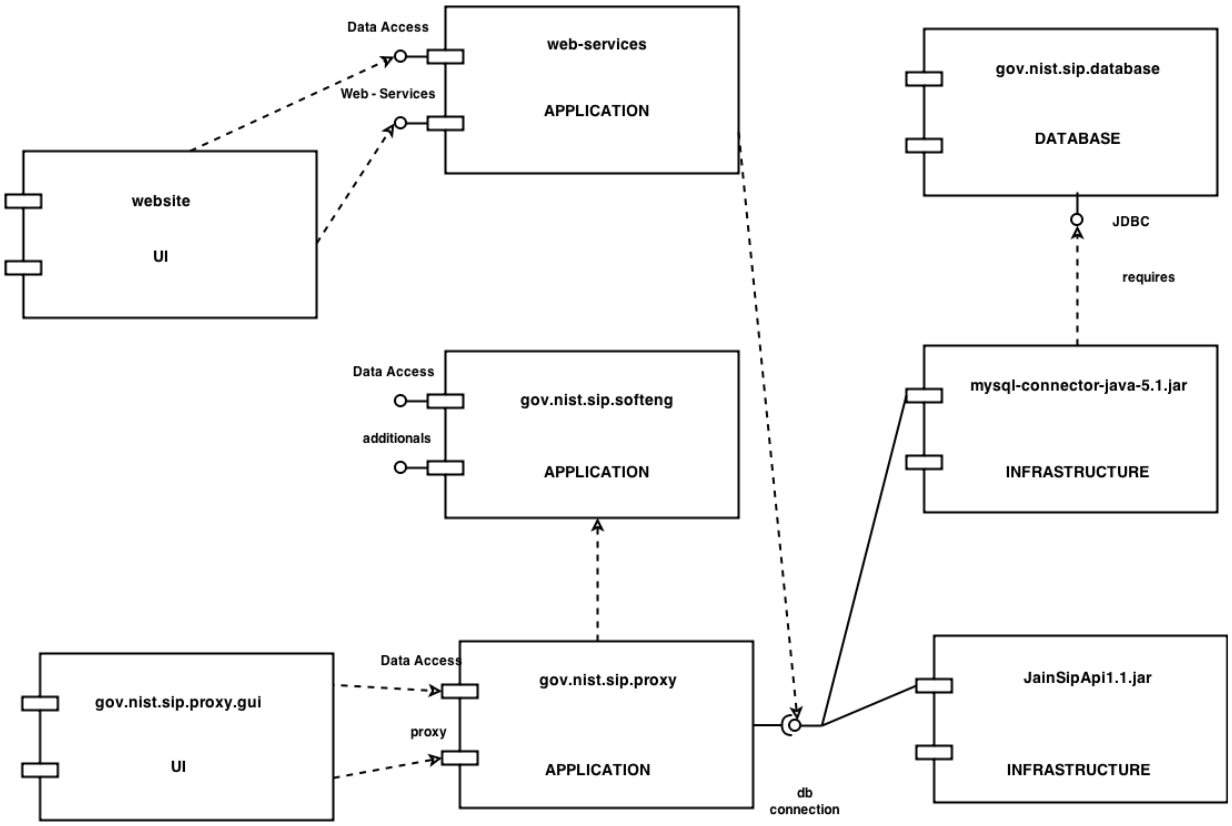
Τέλος, για το billing χρησιμοποιήθηκε το factory method για τα αντικείμενα που υπολογίζουν το κόστος της κλήσης γιατί μας δίνει τη δυνατότητα να δημιουργούμε αντικείμενα χωρίς να καθορίζουμε επακριβώς την κλάση του αντικειμένου που θα δημιουργηθεί.

3 Αρχιτεκτονική

3.1 Ψηφιδικά Διαγράμματα Εφαρμογής

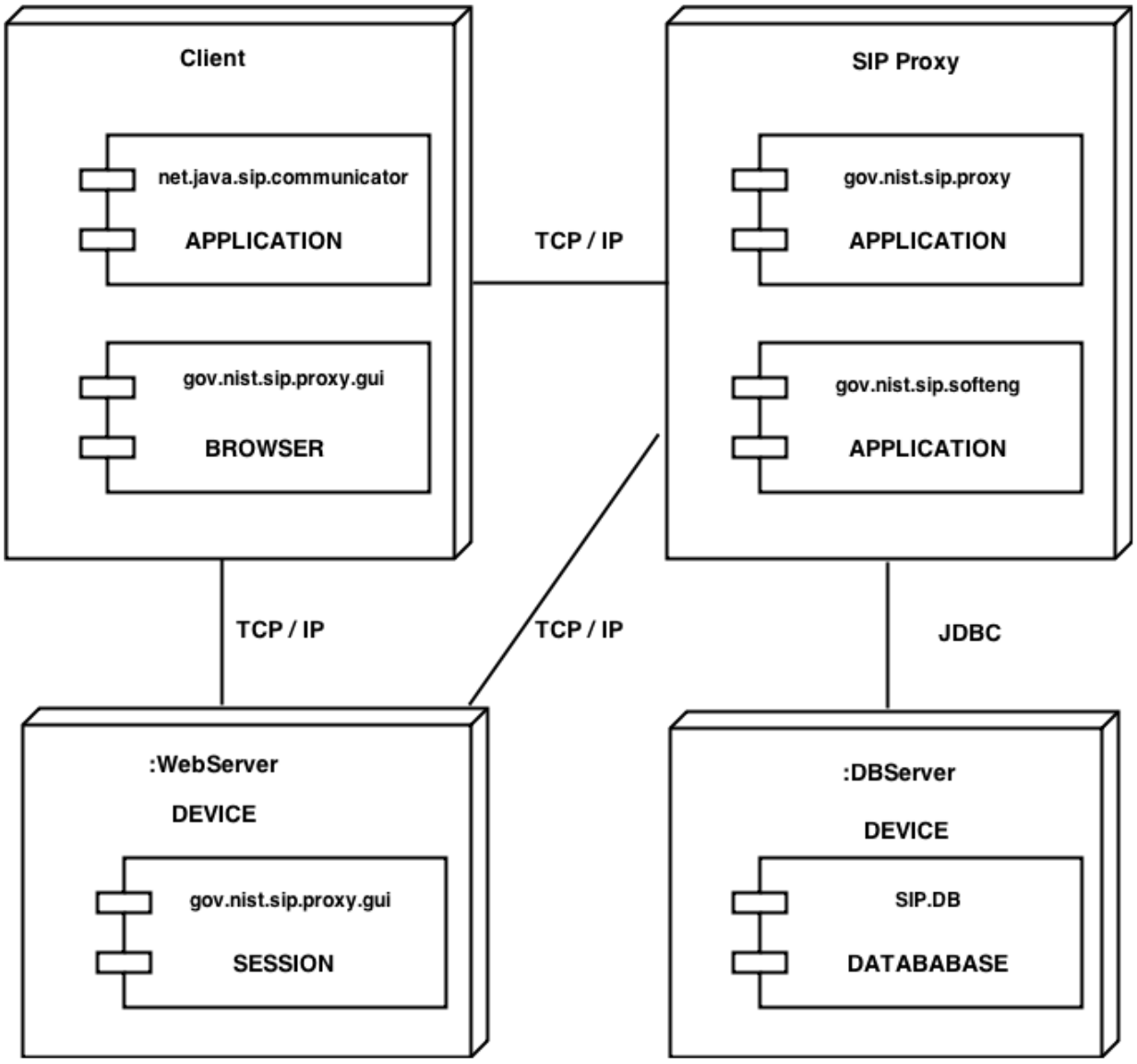


Σχήμα 1: Διάγραμμα για τον SipCommunicator



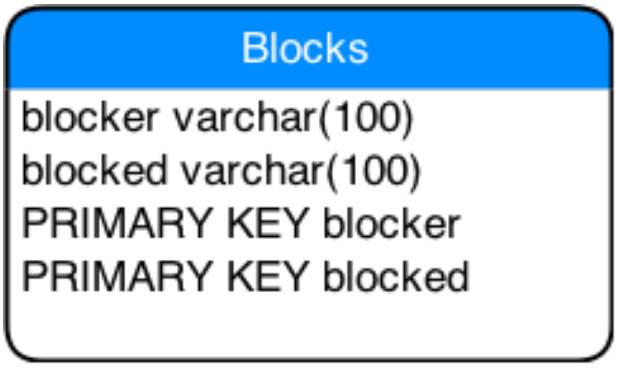
Σχήμα 2: Διάγραμμα για τον SipProxy

3.2 Παραταξιακό Διάγραμμα Εφαρμογής

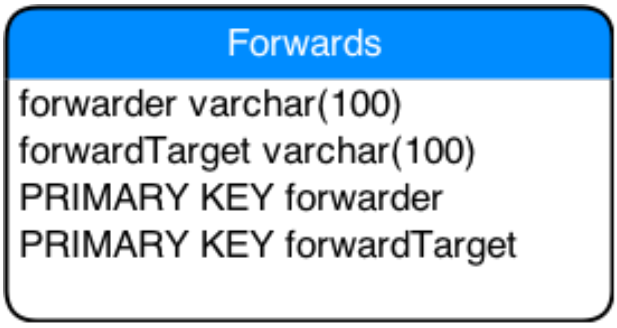


Σχήμα 3: Παραταξιακό διάγραμμα εφαρμογής

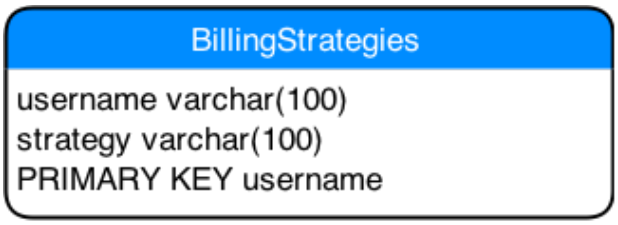
3.3 Πίνακες Οντοτήτων Βάσης



Σχήμα 4: Block db_table

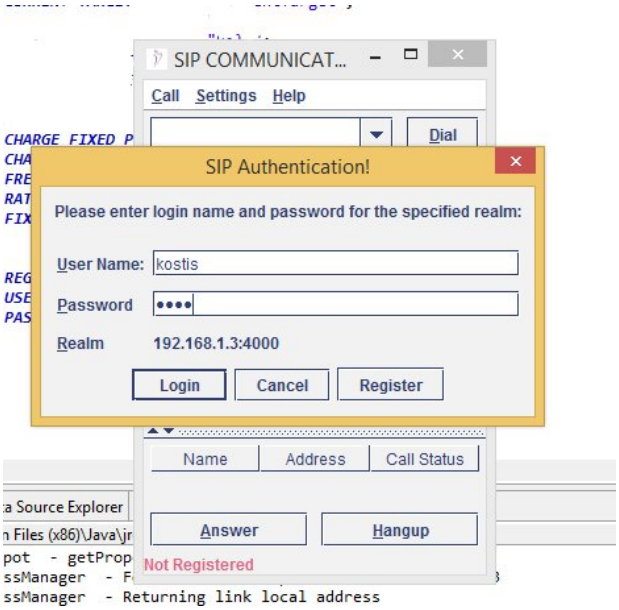


Σχήμα 5: Forward db_table

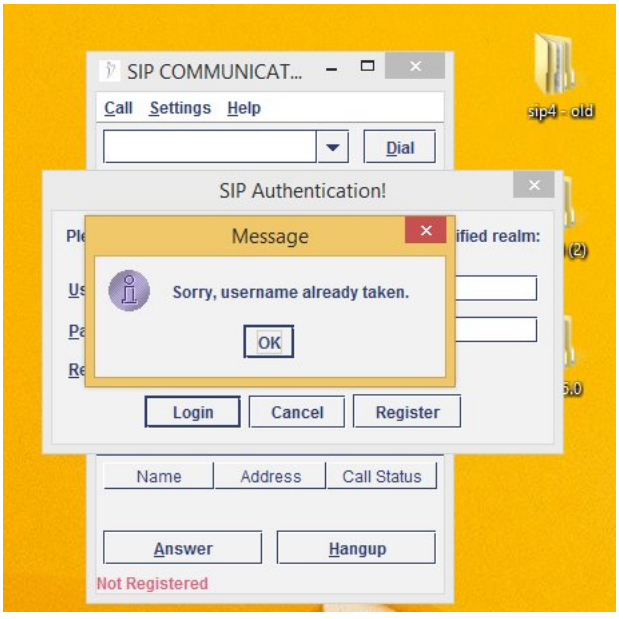


Σχήμα 6: Billing db_table

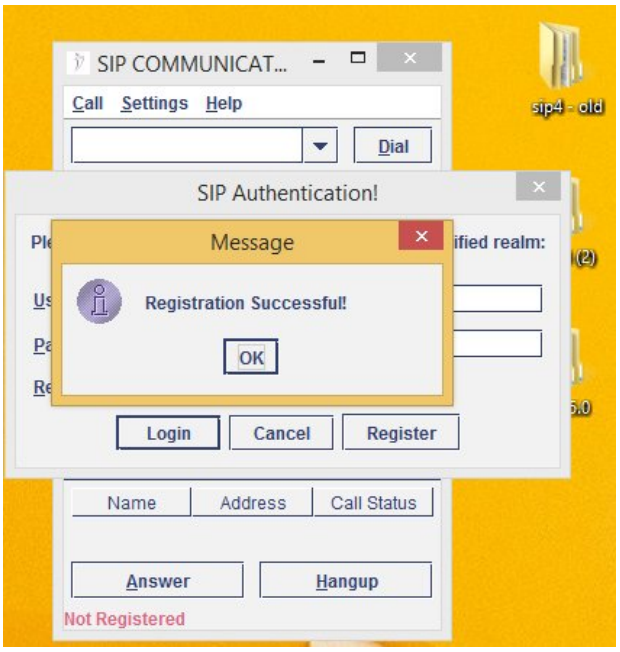
3.4 Screenshots Εφαρμογής



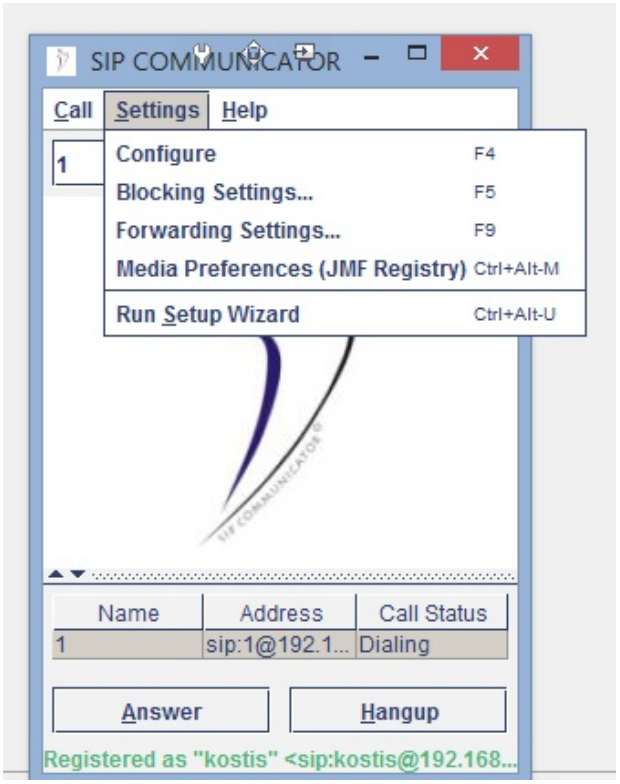
Σχήμα 7: Register Dialog



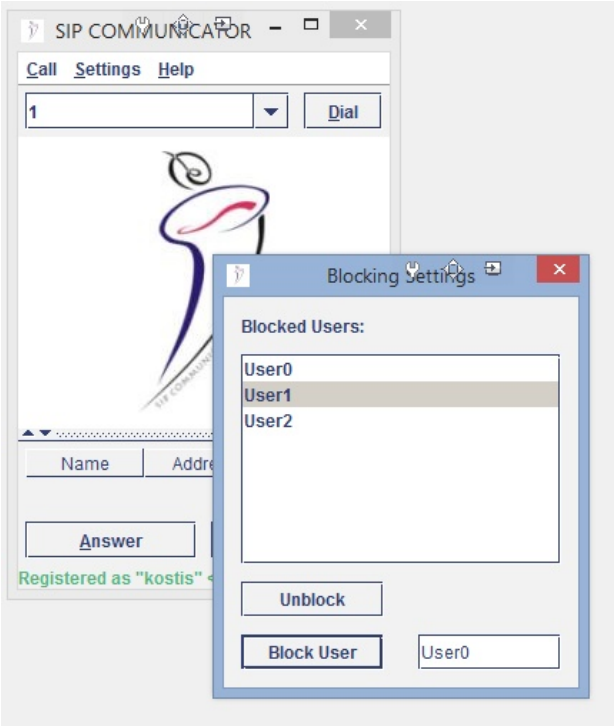
Σχήμα 8: Register Failure Dialog



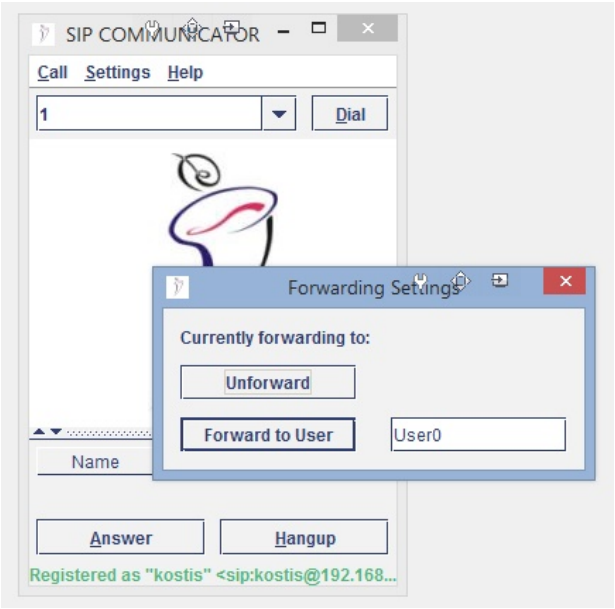
Σχήμα 9: Register Success Dialog



Σχήμα 10: Settings Dialog



Σχήμα 11: Block Dialog



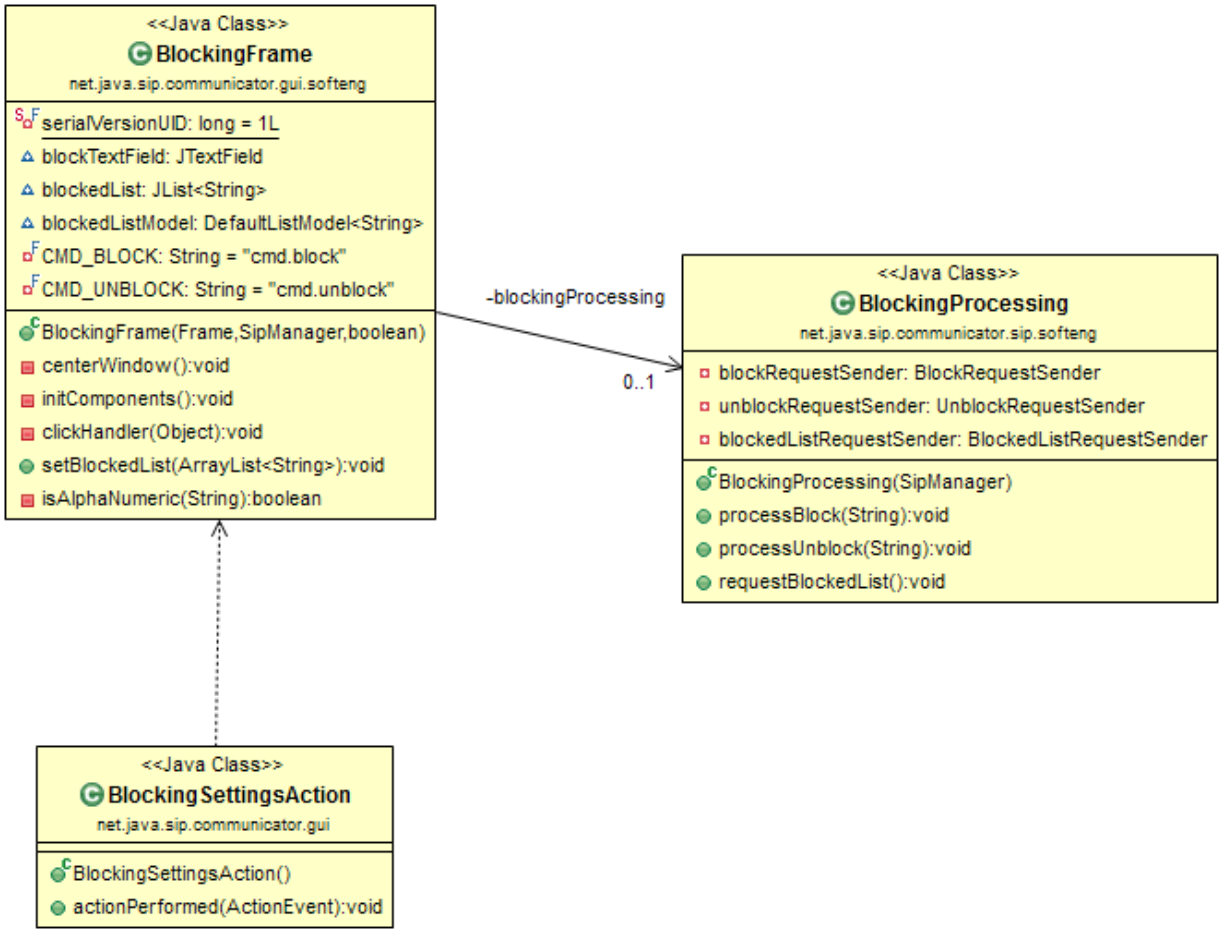
Σχήμα 12: Forward Dialog

4 Λεπτομερή διαγράμματα κλάσεων

4.1 UML Διαγράμματα Κλάσεων

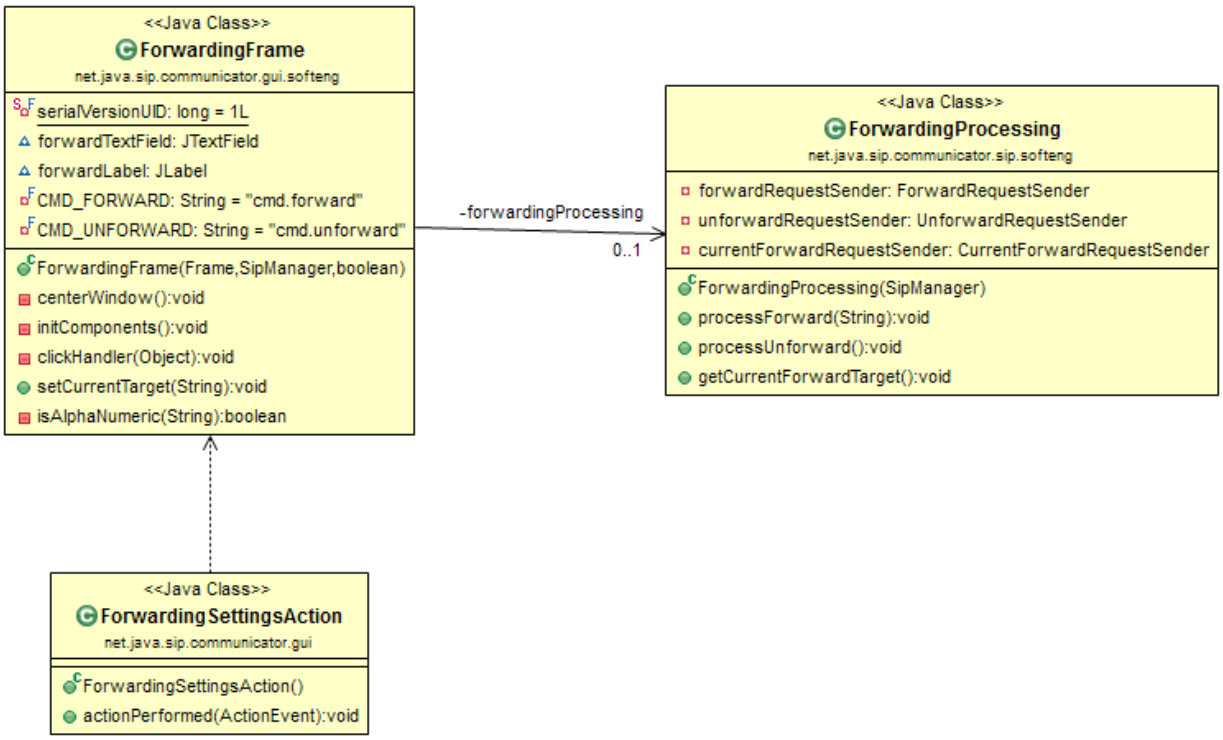
Σε αυτή την ενότητα παρουσιάζουμε τα διαγράμματα των κλάσεων που υλοποιήσαμε, λαμβάνοντας υπόψη σε κάθε διάγραμμα και τις εξαρτήσεις της εκάστοτε κλάσης, από άλλες κλάσεις.

BlockGUI



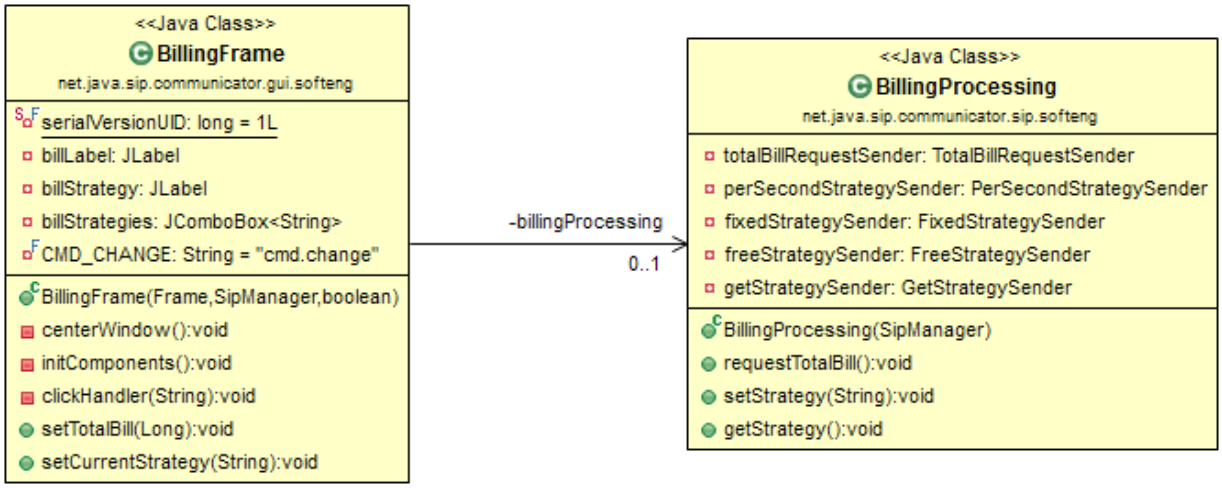
Σχήμα 13: Class Diagram for Blocking GUI

ForwardGUI



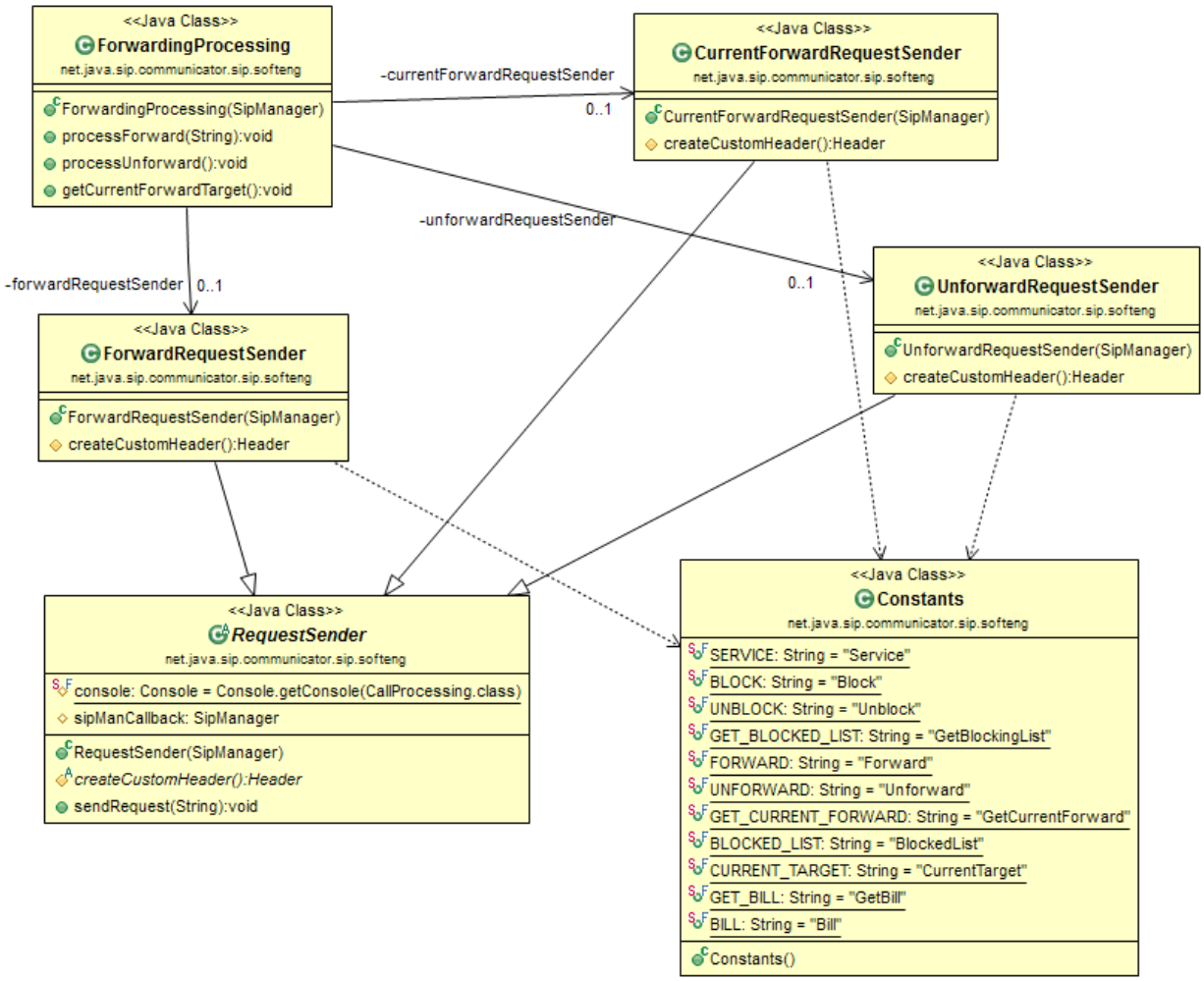
Σχήμα 14: Class Diagram for Forwarding GUI

BillingGUI



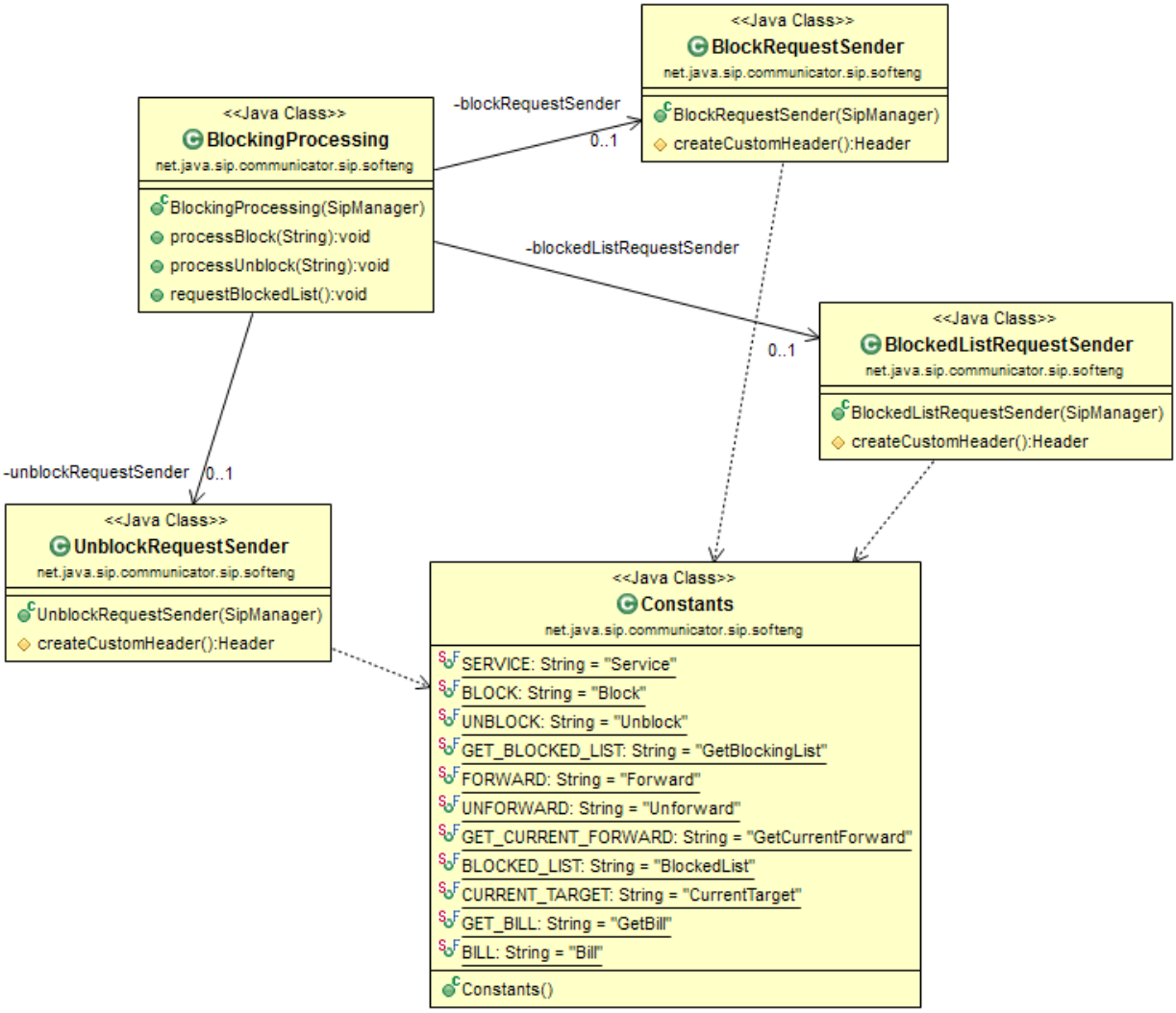
Σχήμα 15: Class Diagram for Billing GUI

ForwardingProcessing



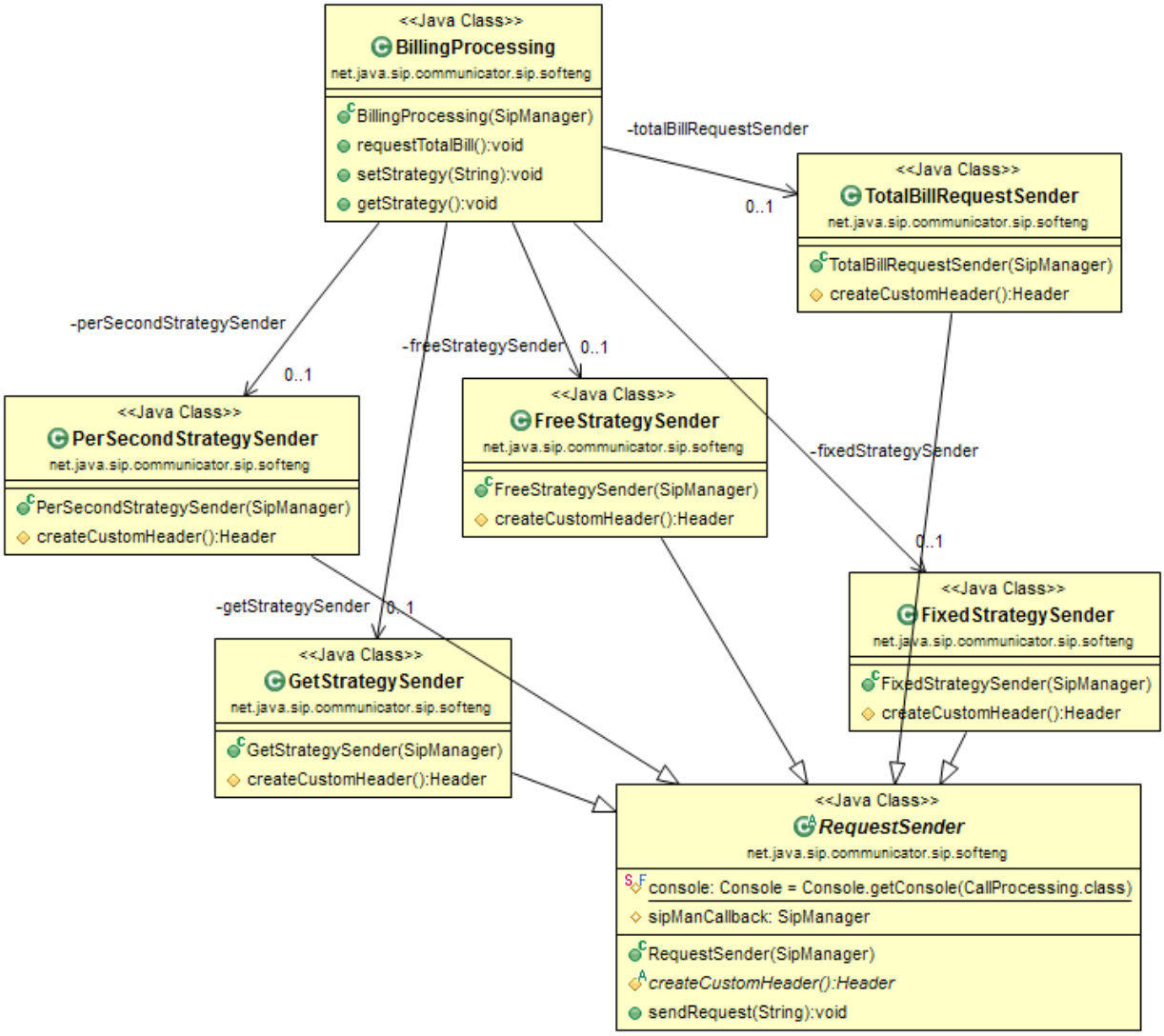
Σχήμα 16: Class Diagram for ForwardingProcessing

BlockingProcessing



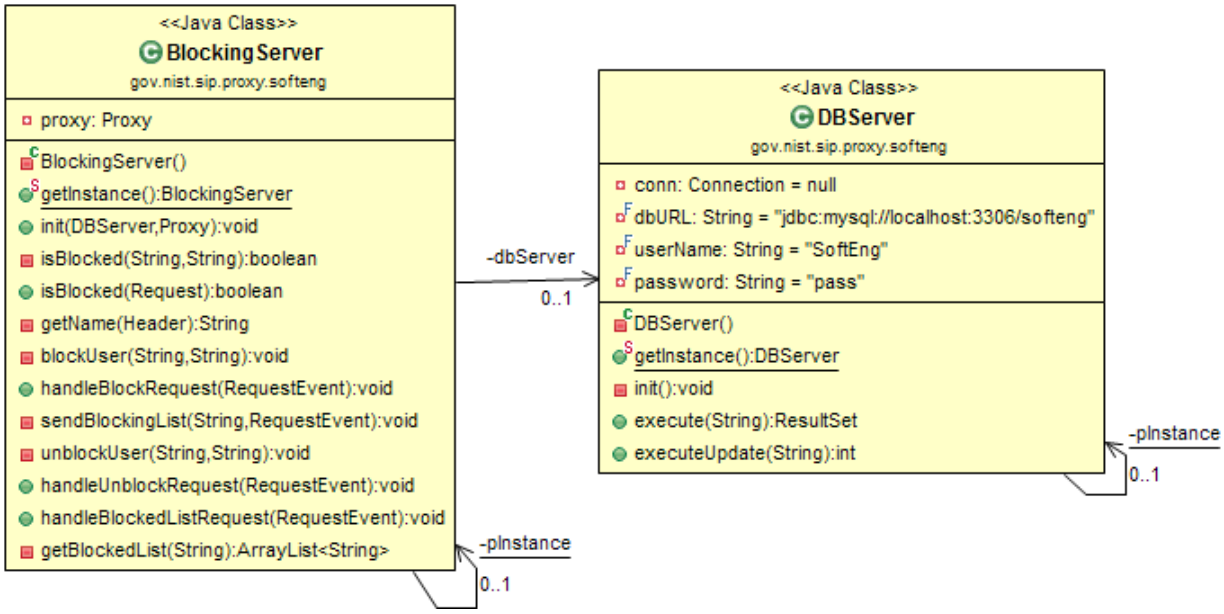
Σχήμα 17: Class Diagram for BlockingProcessing

BillingProcessing



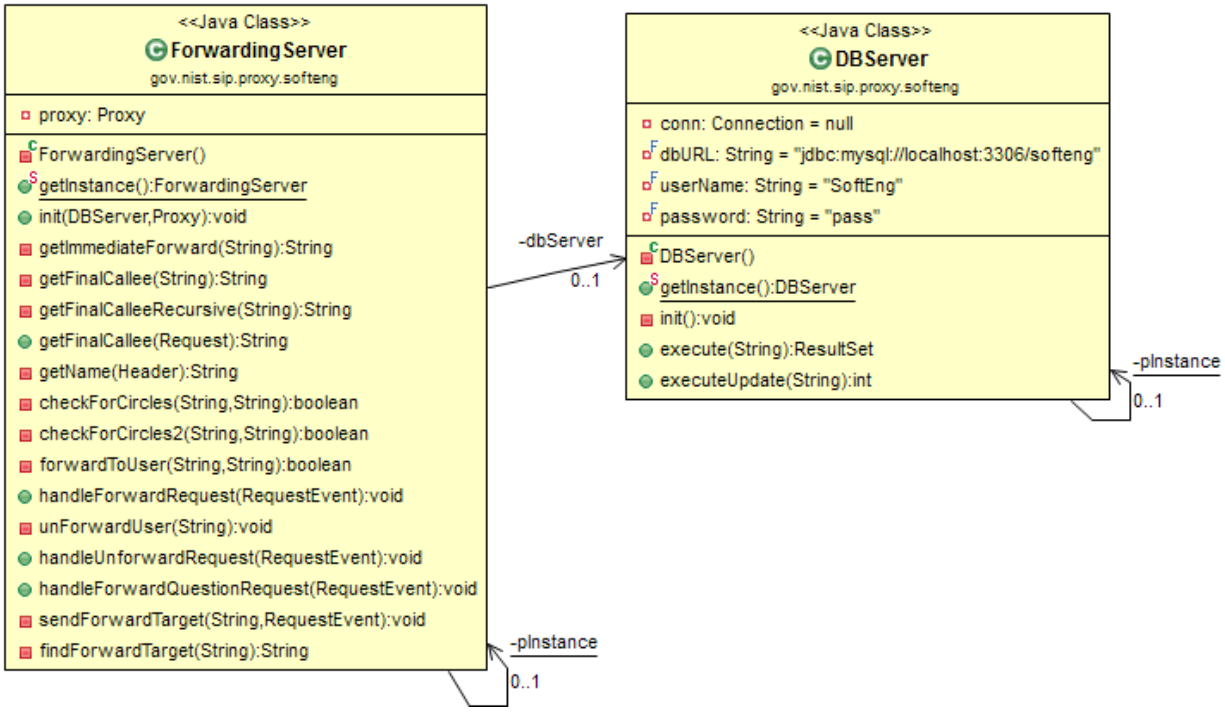
Σχήμα 18: Class Diagram for BillingProcessing

BlockingServer



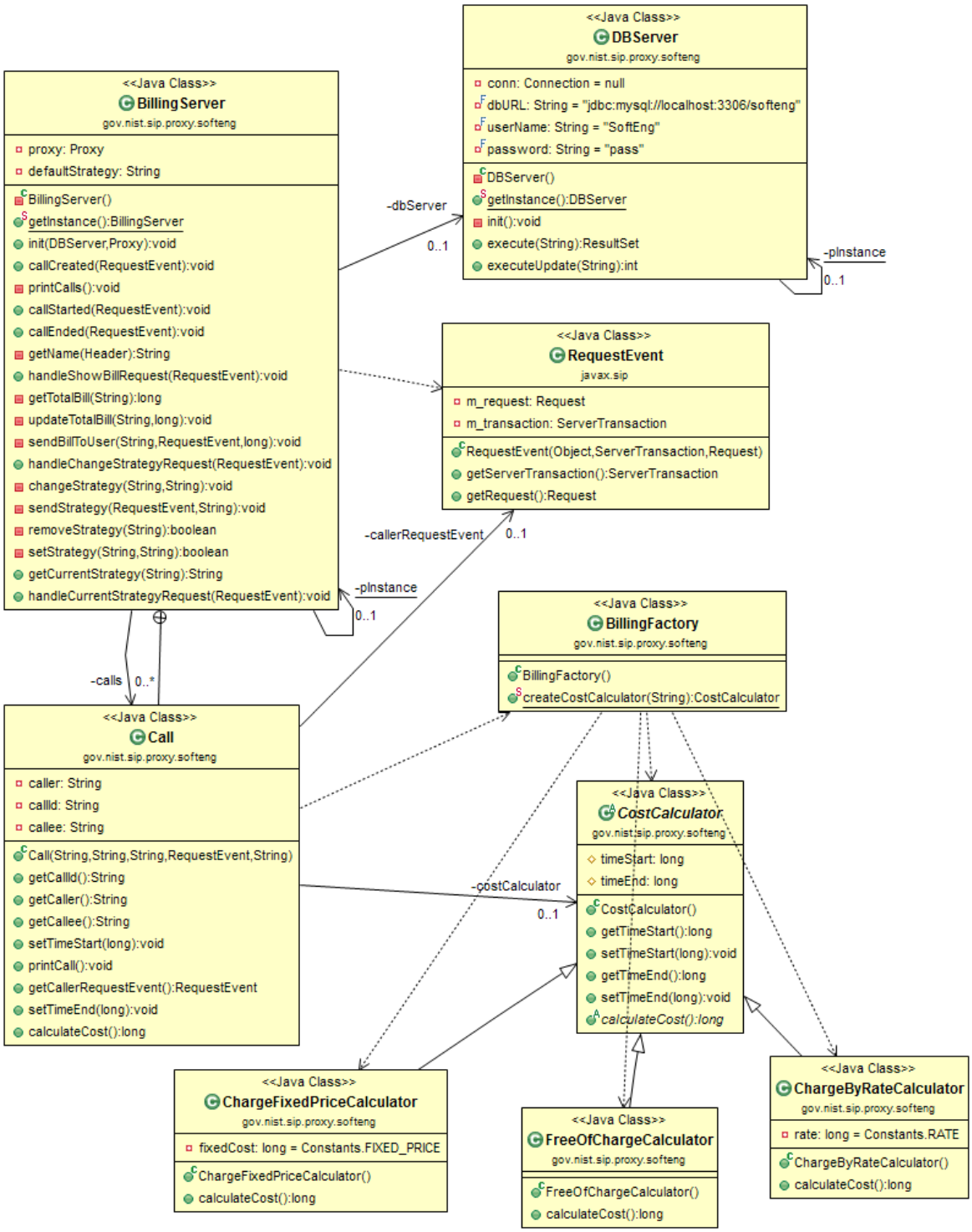
Σχήμα 19: Class Diagram for BlockingServer

ForwardingServer



Σχήμα 20: Class Diagram for ForwardingServer

BillingServer



Σχήμα 21: Class Diagram for BillingServer

4.2 Λεπτομέρειες Μεθόδων

4.2.1 Register

RegisterServer.java

```
package gov.nist.sip.proxy.softeng;

import gov.nist.sip.proxy.Proxy;
import gov.nist.sip.proxy.ProxyDebug;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.ParseException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.sip.RequestEvent;
import javax.sip.ServerTransaction;
import javax.sip.SipException;
import javax.sip.SipProvider;
import javax.sip.header.Header;
import javax.sip.message.MessageFactory;
import javax.sip.message.Request;
import javax.sip.message.Response;

public class RegisterServer {
    private DBServer dbServer;
    private Proxy proxy;

    private static RegisterServer pInstance;

    private RegisterServer()
    {

    }

    public static RegisterServer getInstance()
    {
        if (pInstance == null)
            pInstance = new RegisterServer();

        return pInstance;
    }

    public void init(DBServer dbServer, Proxy proxy)
    {
        this.dbServer = dbServer;
        this.proxy    = proxy;
    }
```

```
public void handleRegisterRequest(RequestEvent requestEvent)
{
    Request request = requestEvent.getRequest();
    Header user = request.getHeader("Username");
    Header pass = request.getHeader("Password");
    String username = null;
    String password = null;
    try {
        username = getValue(user, Constants.USERNAME);
        password = getValue(pass, Constants.PASSWORD);
    } catch (Exception e) {
        ProxyDebug.println("Could not get the username or password from the register request");
        return;
    }
    if (isUsernameInDB(username)) {
        System.out.println("Sending existingUsername response to user: " + username);
        sendExistingUsername(requestEvent);
        return;
    }
    registerUser(username, password);
    sendRegisterOk(requestEvent);
    System.out.println("Sending registerOk response to user: " + username);
}

private String getValue(Header header, String name)
{
    String headerStr = header.toString().trim();
    String value = headerStr.replace(name + ": ", "");
    return value;
}

private boolean registerUser(String username, String password) {
    String query = "Insert into RegisteredUsers values(\"" + username
            + "\", \"" + password + "\");";
    if (dbServer.executeUpdate(query) == -1) {
        System.out.println("Registering User Failed.");
        return false;
    }
    return true;
}

private boolean isUsernameInDB(String username) {
    String query = "Select * from RegisteredUsers where username = \""
            + username + "\"";
    ResultSet res = dbServer.execute(query);
    try {
        if (res.first())
            return true;
    } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        return false;
    }

    private void sendRegisterOk(RequestEvent requestEvent) {
        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        Request request = requestEvent.getRequest();
        MessageFactory messageFactory = proxy.getMessageFactory();
        Response response;
        try {
            response = messageFactory.createResponse(Response.AMBIGUOUS, request);
        } catch (ParseException e) {
            ProxyDebug.println("Failed to create response");
            e.printStackTrace();
            return;
        }
        ServerTransaction serverTransaction = requestEvent.getServerTransaction();
        try {
            if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else
                sipProvider.sendResponse(response);
        } catch (SipException e) {
            ProxyDebug.println("Failed to send RegisterOk response");
            e.printStackTrace();
        }
    }

    public void sendBadCredentials(Request request, SipProvider sipProvider,
            ServerTransaction serverTransaction) {
        MessageFactory messageFactory = proxy.getMessageFactory();
        Response response;
        try {
            response = messageFactory.createResponse(Response.BAD_EXTENSION, request);
        } catch (ParseException e) {
            ProxyDebug.println("Failed to create response");
            e.printStackTrace();
            return;
        }
        try {
            if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else
                sipProvider.sendResponse(response);
        } catch (SipException e) {
            ProxyDebug.println("Failed to send RegisterOk response");
            e.printStackTrace();
        }
    }
```

```
    private void sendExistingUsername(RequestEvent requestEvent) {
        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        Request request = requestEvent.getRequest();
        MessageFactory messageFactory = proxy.getMessageFactory();
        Response response;
        try {
            response = messageFactory.createResponse(Response.BAD_EVENT, request);
        } catch (ParseException e) {
            ProxyDebug.println("Failed to create response");
            e.printStackTrace();
            return;
        }
        ServerTransaction serverTransaction = requestEvent.getServerTransaction();
        try {
            if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else
                sipProvider.sendResponse(response);
        } catch (SipException e) {
            ProxyDebug.println("Failed to send RegisterOk response");
            e.printStackTrace();
        }
    }

    public boolean areCredentialsValid(String username, String password) {
        String query = "Select * from RegisteredUsers where username = \""
                + username + "\"";
        ResultSet res = dbServer.execute(query);
        String pass = "";
        try {
            if (res.first())
                pass = res.getString("password");
            else return false;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        System.out.println("DB pass = " + pass + ", Message pass = " + password);
        return (password.equals(pass));
    }
}
```

Η κλάση αυτή είναι ο server που χρησιμοποιείται όταν κάποιος χρήστης γίνεται register στο σύστημα. Έτσι, αν κάποιος ζητήσει να γίνει register αλλά δεν υπάρχουν τα στοιχεία του, τον ενημερώνουμε με κατάλληλο μήνυμα. Ομοίως, και αν υπάρχει ήδη κάποιος χρήστης με αυτό το όνομα. Αν καταφέρει να γίνει register, τον ενημερώνουμε κατάλληλα.

RegisterRequestProcessing.java

```
package net.java.sip.communicator.sip.softeng;

import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;

public class RegisterRequestProcessing
{
    private RegisterRequestSender registerRequestSender;
    private SipManager sipManCallback;

    public RegisterRequestProcessing(SipManager sipManCallback)
    {
        this.sipManCallback = sipManCallback;
    }

    public void processRegister(String username, String password) throws CommunicationsException
    {
        registerRequestSender = new RegisterRequestSender(sipManCallback, username, password);
        registerRequestSender.sendRequest("dummy");
    }
}
```

RegisterRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.HeaderFactory;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class RegisterRequestSender extends RequestSender
{
    private String username;
    private String password;
    HeaderFactory headerFactory;

    public RegisterRequestSender(SipManager sipManCallback, String username, String password)
    {
        super(sipManCallback);
        this.username = username;
        this.password = password;
        this.headerFactory = sipManCallback.headerFactory;
    }
```

```
    @Override
    protected void addCustomHeaders(Request request) throws ParseException
    {
        // service header
        request.addHeader(headerFactory.createHeader(Constants.SERVICE, Constants.REGISTER));
        // username header
        request.addHeader(headerFactory.createHeader(Constants.USERNAME, username));
        // password header
        request.addHeader(headerFactory.createHeader(Constants.PASSWORD, password));
    }
}
```

Αυτή είναι η abstract class που υλοποιεί τα κοινά μέρη του αλγορίθμου αποστολής μηνυμάτων, σύμφωνα με το template method design pattern.

4.2.2 Blocking

BlockingFrame.java

Η κλάση αυτή υλοποιεί το UI, ή αλλιως το παράθυρο που εμφανίζεται μόλις πατήσουμε το κουμπί Blocking Settings... από την drop down list «Settings». Στο παράθυρο αυτό, όπως βλέπουμε και από τα screenshots έχουμε τη δυνατότητα αρχικά να δούμε όλους τους χρήστες που έχουμε μπλοκάρει, να μπλοκάρουμε κάποιον καινούριο χρήστη ή να ξεμπλοκάρουμε κάποιον από τους χρήστες από τη λίστα των μπλοκαρισμένων. Η ανανέωση της λίστας γίνεται αυτόματα, κάθε φορά που κάνουμε μία νέα ενέργεια. Για να μπλοκάρουμε έναν νέο χρήστη, αρκεί να πληκτρολογήσουμε το όνομά του, ενώ για να ξεμπλοκάρουμε κάποιον, αρκεί να τον επιλέξουμε από τη λίστα των μπλοκαρισμένων. Για το λόγο αυτό, χρησιμοποιούμε τις συναρτήσεις `clickHandler`, η οποία καθορίζει ποια ενέργεια έχει ζητηθεί να γίνει και `setBlockedList` για την εμφάνιση των μπλοκαρισμένων χρηστών.

```
package net.java.sip.communicator.gui.softeng;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;

import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;
import net.java.sip.communicator.sip.softeng.BlockingProcessing;

public class BlockingFrame extends JDialog
{
    private static final long serialVersionUID = 1L;

    JTextField blockTextField;
    JList<String> blockedList;
    DefaultListModel<String> blockedListModel;
    private final String CMD_BLOCK   = "cmd.block";
    private final String CMD_UNBLOCK = "cmd.unblock";
    private BlockingProcessing blockingProcessing;


    /**
     * Creates a new Blocking Settings Dialog
     * @param sipManager
     */
    public BlockingFrame(Frame parent, SipManager sipManager, boolean modal)
    {
        super(parent, modal);
        this.blockingProcessing = sipManager.getBlockingProcessing();
        initComponents();
        pack();
        centerWindow();
    }

    /**
     * Centers the window on the screen.
     */
    private void centerWindow()
    {
        Rectangle screen = new Rectangle(
                Toolkit.getDefaultToolkit().getScreenSize());
        Point center = new Point(
```

```
                (int) screen.getCenterX(), (int) screen.getCenterY());
        Point newLocation = new Point(
                center.x - this.getWidth() / 2, center.y - this.getHeight() / 2);
        if (screen.contains(newLocation.x, newLocation.y,
                this.getWidth(), this.getHeight())) {
            this.setLocation(newLocation);
        }
    }

    /**
     * Creates the contents of the Blocking Settings Dialog
     */
    private void initComponents()
    {
        Container contents = getContentPane();
        contents.setLayout(new BorderLayout());

        setTitle("Blocking Settings");
        getAccessibleContext().setAccessibleDescription("Blocking Settings");
         JPanel centerPane = new JPanel();
        centerPane.setLayout(new GridBagLayout());
        int gridy = 1;

        // label on top of window
        JLabel topLabel = new JLabel("Blocked Users:");

        GridBagConstraints c = new GridBagConstraints();
        c.gridx=0;
        c.gridy=gridy++;
        c.gridwidth = 2;
        c.anchor=GridBagConstraints.WEST;
        c.weighty = 0;
        c.insets=new Insets(12,12,0,12);
        centerPane.add(topLabel, c);

        // list of blocked users
        blockedListModel = new DefaultListModel<String>();
        blockedList = new JList<String>(blockedListModel);
        JScrollPane pane = new JScrollPane(blockedList);

        c = new GridBagConstraints();
        c.gridx = 0;
        c.gridy = gridy++;
        c.gridwidth = 3;
        c.anchor=GridBagConstraints.WEST;
        c.fill = GridBagConstraints.BOTH;
        c.weighty = 1;
        c.insets=new Insets(12,12,0,12);
        centerPane.add(pane, c);
```

```
        // Unblock button
        JButton unblockButton = new JButton();
        unblockButton.setText("Unblock");
        unblockButton.setPreferredSize(new Dimension(100, 25));
        unblockButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                clickHandler(CMD_UNBLOCK);
            }
        });

        c = new GridBagConstraints();
        c.gridx   = 0;
        c.gridy   = gridy++;
        c.anchor  = GridBagConstraints.WEST;
        c.fill    = GridBagConstraints.HORIZONTAL;
        c.weightx = 1;
        c.weighty = 0;
        c.insets  = new Insets(12,12,0,12);
        centerPane.add(unblockButton, c);

        // Block button
        JButton blockButton = new JButton();
        blockButton.setText("Block User");
        blockButton.setPreferredSize(new Dimension(100, 25));
        blockButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                clickHandler(CMD_BLOCK);
            }
        });

        c = new GridBagConstraints();
        c.gridx   = 0;
        c.gridy   = gridy;
        c.anchor  = GridBagConstraints.WEST;
        c.fill    = GridBagConstraints.HORIZONTAL;
        c.weightx = 1;
        c.weighty = 0;
        c.insets  = new Insets(12,12,12,12);
        centerPane.add(blockButton, c);

        // Block user text field
        blockTextField = new JTextField();
        blockTextField.setPreferredSize(new Dimension(100, 25));

        c = new GridBagConstraints();
        c.gridx   = 1;
```

```
        c.gridy   = gridy++;
        c.anchor  = GridBagConstraints.WEST;
        c.fill    = GridBagConstraints.HORIZONTAL;
        c.weightx = 1;
        c.weighty = 0;
        c.insets  = new Insets(12,12,12,12);
        centerPane.add(blockTextField, c);

        contents.add(centerPane, BorderLayout.CENTER);
        getRootPane().setDefaultButton(blockButton);
    }

    /**
     * Handles a click event.
     * If the blocked user list is updated as a result the proxy will
      * respond with the new list asynchronously.
     *
     * @param actionCommand the action performed by the user
     */
    private void clickHandler(Object actionCommand)
    {
        System.out.println("Received command: "+actionCommand+"\tText =
        "+blockTextField.getText());

        if (actionCommand.equals(CMD_BLOCK)) {
            String blocked = blockTextField.getText();
            if (blocked == null || "".equals(blocked)) {
                JOptionPane.showMessageDialog(this,
                        "You must provide the username of the user to be blocked.");
                return;
            } else if (!isAlphaNumeric(blocked)) {
                JOptionPane.showMessageDialog(this,
                        "The username must be alphanumeric");
                return;
            }
            try {
                blockingProcessing.processBlock(blocked);
            } catch (CommunicationsException e) {
                JOptionPane.showMessageDialog(this,
                        "Unable to send the block request. (T__T)");
            }
        } else if (actionCommand.equals(CMD_UNBLOCK)) {
            String unblocked = blockedList.getSelectedValue();
            if (unblocked == null || "".equals(unblocked)) {
                JOptionPane.showMessageDialog(this,
                        "Please select the user to be unblocked.");
                return;
            }
            try {
                blockingProcessing.processUnblock(unblocked);
```

```
            } catch (CommunicationsException e) {
                JOptionPane.showMessageDialog(this,
                        "Unable to process block request. (T__T)");
            }
        }
    }

    /**
     * Sets the list of blocked users
     *
     * @param blockedUsers the new list of blocked users to display
     */
    public void setBlockedList(ArrayList<String> blockedUsers)
    {
        blockedListModel.clear();
        for (String user : blockedUsers)
            blockedListModel.addElement(user);
    }

    // source: stackoverflow.org
    private boolean isAlphaNumeric(String s){
        String pattern= "^[a-zA-Z0-9]*$";
            if(s.matches(pattern)){
                return true;
            }
            return false;
    }
}
```

BlockingProcessing.java

```
package net.java.sip.communicator.sip.softeng;

import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;

public class BlockingProcessing {

    private BlockRequestSender blockRequestSender;
    private UnblockRequestSender unblockRequestSender;
    private BlockedListRequestSender blockedListRequestSender;

    public BlockingProcessing(SipManager sipManCallback)
    {
        blockRequestSender          = new BlockRequestSender(sipManCallback);
        unblockRequestSender      = new UnblockRequestSender(sipManCallback);
        blockedListRequestSender = new BlockedListRequestSender(sipManCallback);
    }
```

```
    public void processBlock(String blocked) throws CommunicationsException
    {
        blockRequestSender.sendRequest(blocked);
    }

    public void processUnblock(String target) throws CommunicationsException
    {
        unblockRequestSender.sendRequest(target);
    }

    public void requestBlockedList() throws CommunicationsException
    {
        blockedListRequestSender.sendRequest("dummy"); // dummy target
    }
}
```

Η κλάση BlockingProcessing είναι η κλάση που αναλαμβάνει να χειριστεί τα μηνύματα που έρχονται από τον χρήστη. Έτσι, ανάλογα με το αν ο χρήστης έχει ζητήσει να μπλοκάρει κάποιον άλλον χρήστη, αν έχει ζητήσει να ξε-μπλοκάρει κάποιον ή αν έχει ζητήσει τη λίστα με τους μπλοκαρισμένους χρήστες, η BlockingProcessing αναλαμβάνει να προωθήσει το αίτημα στις κατάλληλες υποκλάσεις.

BlockedListRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class BlockedListRequestSender extends RequestSender
{
    public BlockedListRequestSender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected void addCustomHeaders(Request request) throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.GET_BLOCKED_LIST;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
```

```
    }
}
```

BlockRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class BlockRequestSender extends RequestSender {

    public BlockRequestSender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected void addCustomHeaders(Request request) throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.BLOCK;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
    }
}
```

UnblockRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class UnblockRequestSender extends RequestSender
{
    public UnblockRequestSender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }
```

```
    @Override
    protected void addCustomHeaders(Request request) throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.UNBLOCK;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
    }
}
```

Κάθε μία από τις παραπάνω κλάσεις, φτιάχνει κατάλληλα το Header έτσι ώστε να προωθηθεί στον server το αίτημα του χρήστη.

Το BlockingFrame καλείται από την κλάση GuiManager όταν ο χρήστης πατήσει το κατάλληλο κουμπί.

Η χειρισμός των requests που στέλνονται καθώς ο χρήστης πατάει διάφορα κουμπιά στο παράθυρο αυτό, γίνεται μέσω της κλάσης `BlockingServer.java`, η οποία βρίσκεται μέσα στον proxy και είναι αυτή που αναλαμβάνει να μετατρέψει τα αιτήματα του χρήστη σε mysql statements και να επικοινωνήσει με τη βάση. Για το λόγο αυτό, χρειάζεται κάθε request του communicator να προωθείται στον proxy.

BlockingServer.java

```
package gov.nist.sip.proxy.softeng;

import gov.nist.sip.proxy.Proxy;
import gov.nist.sip.proxy.ProxyDebug;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.sip.RequestEvent;
import javax.sip.ServerTransaction;
import javax.sip.SipException;
import javax.sip.SipProvider;
import javax.sip.header.Header;
import javax.sip.header.HeaderFactory;
import javax.sip.message.MessageFactory;
import javax.sip.message.Request;
import javax.sip.message.Response;

public class BlockingServer {
```

```
    private DBServer dbServer;
    private Proxy    proxy;

    // TODO kostis 25-3-2015
    private static BlockingServer pInstance;

    private BlockingServer() {}

    public static BlockingServer getInstance()
    {
        if (pInstance == null)
            pInstance = new BlockingServer();

        return pInstance;
    }

    public void init(DBServer dbServer, Proxy proxy)
    {
        this.dbServer = dbServer;
        this.proxy    = proxy;
    }
    // TODO end

    private boolean isBlocked(String caller, String callee)
    {
        String query = "Select * from blocks where blocker = \"" + callee +
                       "\" and blocked = \"" + caller + "\";";
        ResultSet res = dbServer.execute(query);
        boolean blocked = false;
        try {
            blocked = res.first();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return blocked;
    }

    public boolean isBlocked(Request request)
    {
        Header from = request.getHeader("From");
        Header to   = request.getHeader("To");
        String caller = null;
        String callee = null;
        try {
            caller = getName(from);
            callee = getName(to);
        } catch (Exception e) {
            return true;
        }
        return isBlocked(caller, callee);
```

```
    }

    private String getName(Header header)
    {
        String pattern = "<sip:([^;@]*)[@;]";
        Pattern r = Pattern.compile(pattern);
        Matcher m = r.matcher(header.toString());
        m.find();
        return m.group(1);
    }

    private void blockUser(String blocker, String blocked)
    {
        String query = "Insert into blocks values(\"" + blocker +
                       "\", \"" + blocked + "\");";
        if (dbServer.executeUpdate(query) == -1)
            System.out.println("Insert Query Failed");
    }

    public void handleBlockRequest(RequestEvent requestEvent)
    {
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        Header to   = request.getHeader("To");
        String blocker = null;
        String blocked = null;
        try {
            blocker = getName(from);
            blocked = getName(to);
        } catch (Exception e) {
            return;
        }
        blockUser(blocker, blocked);
        sendBlockingList(blocker, requestEvent);
    }

    private void sendBlockingList(String blocker, RequestEvent requestEvent)
    {
        // Construct the blocked users list string
        ArrayList<String> blockedUsers = getBlockedList(blocker);
        String blockedList = "";
        for (String user : blockedUsers)
            blockedList += user + "%";
        if (!blockedList.equals("")) // remove the ;; at the end
            blockedList = blockedList.substring(0, blockedList.length() -1);

        System.out.println("Sending blocking list: " + blockedList);

        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        Request request = requestEvent.getRequest();
```

```
        HeaderFactory headerFactory = proxy.getHeaderFactory();
        MessageFactory messageFactory = proxy.getMessageFactory();
        Response response;
        try {
            response = messageFactory.createResponse(Response.OK, request);
        } catch (ParseException e) {
            ProxyDebug.println("Failed to create response");
            e.printStackTrace();
            return;
        }

        // Blocking List Service header
        String headerName  = Constants.SERVICE;
        String headerValue = Constants.GET_BLOCKED_LIST;
        Header serviceHeader;
        try {
            serviceHeader = headerFactory.createHeader(headerName, headerValue);
        } catch (ParseException e) {
            ProxyDebug.println("An unexpected error occurred while constructing the Service Header");
            e.printStackTrace();
            return;
        }
        response.addHeader(serviceHeader);

        // Blocking List Content header
        String blockedListName = Constants.BLOCKED_LIST;
        Header blockedListHeader;
        try {
            blockedListHeader = headerFactory.createHeader(blockedListName, blockedList);
        } catch (ParseException e) {
            ProxyDebug.println("An unexpected error occurred while constructing the Service Header");
            e.printStackTrace();
            return;
        }
        response.addHeader(blockedListHeader);

        ServerTransaction serverTransaction = requestEvent.getServerTransaction();
        try {
            if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else
                sipProvider.sendResponse(response);
        } catch (SipException e) {
            ProxyDebug.println("Failed to send blockingList response");
            e.printStackTrace();
        }
    }

    private void unblockUser(String blocker, String blocked)
    {
```

```
        String query = "delete from blocks where blocker = \"" + blocker +
                       "\" and blocked = \"" + blocked + "\";";
        if (dbServer.executeUpdate(query) == -1)
            System.out.println("Delete Query Failed.");
    }

    public void handleUnblockRequest(RequestEvent requestEvent)
    {
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        Header to   = request.getHeader("To");
        String blocker = null;
        String blocked = null;
        try {
            blocker = getName(from);
            blocked = getName(to);
        } catch (Exception e) {
            return;
        }
        unblockUser(blocker, blocked);
        sendBlockingList(blocker, requestEvent);
    }

    public void handleBlockedListRequest(RequestEvent requestEvent)
    {
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        String blocker = null;
        try {
            blocker = getName(from);
        } catch (Exception e) {
            ProxyDebug.println("Could not get the user name from the request");
            return;
        }
        sendBlockingList(blocker, requestEvent);
    }

    private ArrayList<String> getBlockedList(String blocker)
    {
        String query = "Select blocked from blocks where blocker = \"" + blocker + "\";";
        ResultSet res = dbServer.execute(query);
        ArrayList<String> results = new ArrayList<String>();
        try {
            while (res.next()) {
                String blocked = res.getString("blocked");
                results.add(blocked);
            }
        } catch (Exception e) {
            return null;
        }
```

```
        return results;
    }
}
```

Με τη μέθοδο `isBlocked` ελέγχουμε εάν ένας χρήστης B είναι μπλοκαρισμένος από τον χρήστη A. Αυτή η μέθοδος είναι απαραίτητη προκειμένου να ελέγχουμε ποιοι χρήστες είναι μπλοκαρισμένοι και να μην τους προωθούμε πλέον κλήσεις (θα το δούμε αυτό σε μεταγενέστερη κλάση). Η `blockUser` προσθέτει στη βάση δεδομένων ένα νέο ζεύγος 'blocker,blocked' ενώ η `unblockUser` πηγαίνει στη βάση και διαγράφει το ζεύγος. Οι συναρτήσεις `handleBlockRequest, handleUnblockRequest` είναι για να μπορούμε να διαχειριστούμε στον `proxy` το request που μας έρχεται από τον communicator. Τέλος, η `getBlockedList` επιστρέφει μία λίστα με τους μπλοκαρισμένους χρήστες.

4.2.3 Forwarding

ForwardingFrame.java

```
package net.java.sip.communicator.gui.softeng;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;
import net.java.sip.communicator.sip.softeng.ForwardingProcessing;
```

```
public class ForwardingFrame extends JDialog
{
    private static final long serialVersionUID = 1L;

    // TODO kostis 24-1-2015
    JTextField forwardTextField;
    JLabel forwardLabel;
    private final String CMD_FORWARD   = "cmd.forward";
    private final String CMD_UNFORWARD = "cmd.unforward";
    private ForwardingProcessing forwardingProcessing;
    // TODO kostis

    /**
     * Creates a new Forwarding Settings Dialog
     * @param sipManager
     */
    public ForwardingFrame(Frame parent, SipManager sipManager, boolean modal)
    {
        super(parent, modal);
        this.forwardingProcessing = sipManager.getForwardingProcessing();
        initComponents();
        pack();
        centerWindow();
    }

    /**
     * Centers the window on the screen.
     */
    private void centerWindow()
    {
        Rectangle screen = new Rectangle(
                Toolkit.getDefaultToolkit().getScreenSize());
        Point center = new Point(
                (int) screen.getCenterX(), (int) screen.getCenterY());
        Point newLocation = new Point(
                center.x - this.getWidth() / 2, center.y - this.getHeight() / 2);
        if (screen.contains(newLocation.x, newLocation.y,
                this.getWidth(), this.getHeight())) {
            this.setLocation(newLocation);
        }
    }

    /**
     * Creates the contents of the Forwarding Settings Dialog
     */
    private void initComponents()
    {
        Container contents = getContentPane();
        contents.setLayout(new BorderLayout());
```

```
setTitle("Forwarding Settings");
getAccessibleContext().setAccessibleDescription("Forwarding Settings");
 JPanel centerPane = new JPanel();
centerPane.setLayout(new GridBagLayout());
int gridy = 1;

// label on top of window
// TODO kostis 24-1-2015
forwardLabel = new JLabel("Currently forwarding to:");
GridBagConstraints c = new GridBagConstraints();
c.gridx=0;
c.gridy=gridy++;
c.gridwidth = 2;
c.anchor=GridBagConstraints.WEST;
c.weighty = 0;
c.insets=new Insets(12,12,0,12);
centerPane.add(forwardLabel, c);
// TODO end

// Unforward button
JButton unforwardButton = new JButton();
unforwardButton.setText("Unforward");
unforwardButton.setPreferredSize(new Dimension(125, 25));
unforwardButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        clickHandler(CMD_UNFORWARD);
    }
});

c = new GridBagConstraints();
c.gridx   = 0;
c.gridy   = gridy++;
c.anchor  = GridBagConstraints.WEST;
c.fill    = GridBagConstraints.HORIZONTAL;
c.weightx = 1;
c.weighty = 0;
c.insets  = new Insets(12,12,0,12);
centerPane.add(unforwardButton, c);

// Forward button
JButton forwardButton = new JButton();
forwardButton.setText("Forward to User");
forwardButton.setPreferredSize(new Dimension(125, 25));
forwardButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        clickHandler(CMD_FORWARD);
```

```
            }
        });

        c = new GridBagConstraints();
        c.gridx   = 0;
        c.gridy   = gridy;
        c.anchor  = GridBagConstraints.WEST;
        c.fill    = GridBagConstraints.HORIZONTAL;
        c.weightx = 1;
        c.weighty = 0;
        c.insets  = new Insets(12,12,12,12);
        centerPane.add(forwardButton, c);

        // Forward to user text field
        forwardTextField = new JTextField();
        forwardTextField.setPreferredSize(new Dimension(125, 25));

        c = new GridBagConstraints();
        c.gridx   = 1;
        c.gridy   = gridy++;
        c.anchor  = GridBagConstraints.WEST;
        c.fill    = GridBagConstraints.HORIZONTAL;
        c.weightx = 1;
        c.weighty = 0;
        c.insets  = new Insets(12,12,12,12);
        centerPane.add(forwardTextField, c);

        contents.add(centerPane, BorderLayout.CENTER);
        getRootPane().setDefaultButton(forwardButton);
    }

    /**
     * Handles a click event.
     * If the blocked user list is updated as a result the proxy will
      * respond with the new list asynchronously.
     *
     * @param actionCommand the action performed by the user
     */
    private void clickHandler(Object actionCommand)
    {
        System.out.println("Received command: "+actionCommand+
            "\tText = "+forwardTextField.getText());

        if (actionCommand.equals(CMD_FORWARD)) {
            String forwarding = forwardTextField.getText();
            if (forwarding == null || "".equals(forwarding)) {
                JOptionPane.showMessageDialog(this,
                    "You must provide the username of the user to forward your calls.");
                return;
            }
```

```
            else if (!isAlphaNumeric(forwarding)) {
                JOptionPane.showMessageDialog(this,
                    "The username must be Alphanumeric");
                return;
            }
            try {
                forwardingProcessing.processForward(forwarding);
            } catch (CommunicationsException e) {
                JOptionPane.showMessageDialog(this,
                    "Unable to send the forward request. (T__T)");
            }
        } else if (actionCommand.equals(CMD_UNFORWARD)) {
            try {
                forwardingProcessing.processUnforward();
            } catch (CommunicationsException e) {
                JOptionPane.showMessageDialog(this,
                    "Unable to process unforward request. (T__T)");
            }
        }
    }

    // TODO kostis 24-1-2015
    /**
     * Sets the displayed forwarding target to the given string
     *
     * @param target the target the user is currently forwarding to
     */
    public void setCurrentTarget(String target)
    {
        forwardLabel.setText("Currently forwarding to:    " + target);
    }
    // TODO end

    // source: stackoverflow.org
    private boolean isAlphaNumeric(String s){
        String pattern= "^[a-zA-Z0-9]*$";
            if(s.matches(pattern)){
                return true;
            }
            return false;
    }
}
```

Η κλάση αυτή υλοποιεί το UI, ή αλλιως το παράθυρο που εμφανίζεται μόλις πατήσουμε το κουμπί Forwarding Settings... από την drop down list «Settings». Στο παράθυρο αυτό, όπως βλέπουμε και από τα screenshots έχουμε τη δυνατότητα αρχικά να δούμε πού έχουμε προωθήσει τις κλήσεις μας, αλλά, να προσθέσουμε κάποιον χρήστη στον οποίο να

προωθούμε, καθώς και να αφαιρέσουμε την προώθηση κλήσεων. Για να προωθήσουμε ή να άρουμε την προώθηση σε κάποιον χρήστη πληκτρολογούμε το όνομά του. Η ανανέωση του ονόματος στο οποίο προωθούμε γίνεται αυτόματα, κάθε φορά που κάνουμε μία νέα ενέργεια.

ForwardingProcessing.java

```
package net.java.sip.communicator.sip.softeng;

import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;

public class ForwardingProcessing
{
    private ForwardRequestSender forwardRequestSender;
    private UnforwardRequestSender unforwardRequestSender;
    private CurrentForwardRequestSender currentForwardRequestSender;

    public ForwardingProcessing(SipManager sipManCallback)
    {
        forwardRequestSender         = new ForwardRequestSender(sipManCallback);
        unforwardRequestSender         = new UnforwardRequestSender(sipManCallback);
        currentForwardRequestSender = new CurrentForwardRequestSender(sipManCallback);
    }

    public void processForward(String target) throws CommunicationsException
    {
        forwardRequestSender.sendRequest(target);
    }

    public void processUnforward() throws CommunicationsException
    {
        unforwardRequestSender.sendRequest("dummy"); // dummy target
    }

    public void getCurrentForwardTarget() throws CommunicationsException
    {
        currentForwardRequestSender.sendRequest("dummy"); // dummy target
    }
}
```

Η κλάση ForwardingProcessing είναι η κλάση που αναλαμβάνει να χειριστεί τα μηνύματα που έρχονται από τον χρήστη. Έτσι, ανάλογα με το αν ο χρήστης έχει ζητήσει να προωθήσει σε κάποιον άλλον χρήστη, αν έχει ζητήσει να άρει την προώθηση από κάποιον καθώς και επίσης προκειμένου να προβάλει τον χρήστη στον οποίο προωθούμε αυτή τη στιγμή, η ForwardingProcessing αναλαμβάνει να προωθήσει το αίτημα στις κατάλληλες

υποκλάσεις.

CurrentForwardRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class CurrentForwardRequestSender extends RequestSender
{
    public CurrentForwardRequestSender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected void addCustomHeaders(Request request) throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.GET_CURRENT_FORWARD;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
    }
}
```

ForwardRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class ForwardRequestSender extends RequestSender
{
    public ForwardRequestSender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected void addCustomHeaders(Request request) throws ParseException
```

```
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.FORWARD;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
    }
}
```

UnforwardRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class UnforwardRequestSender extends RequestSender
{
    public UnforwardRequestSender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected void addCustomHeaders(Request request) throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.UNFORWARD;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
    }
}
```

Κάθε μία από τις παραπάνω κλάσεις, φτιάχνει κατάλληλα το Header έτσι ώστε να προωθηθεί στον server το αίτημα του χρήστη.

ForwardingServer.java

```
package gov.nist.sip.proxy.softeng;

import gov.nist.sip.proxy.Proxy;
import gov.nist.sip.proxy.ProxyDebug;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.ParseException;
```

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.sip.RequestEvent;
import javax.sip.ServerTransaction;
import javax.sip.SipException;
import javax.sip.SipProvider;
import javax.sip.header.Header;
import javax.sip.header.HeaderFactory;
import javax.sip.message.MessageFactory;
import javax.sip.message.Request;
import javax.sip.message.Response;

public class ForwardingServer
{
        // TODO kostis 25-3-2015
        private DBServer dbServer;
        private Proxy    proxy;

        private static ForwardingServer pInstance;

        private ForwardingServer() {}

        public static ForwardingServer getInstance()
        {
                if (pInstance == null)
                        pInstance = new ForwardingServer();

                return pInstance;
        }

        public void init(DBServer dbServer, Proxy proxy)
        {
                this.dbServer = dbServer;
                this.proxy    = proxy;
        }

        /*
         * Returns the name of the immediate target the callee is forwarding to,
         * null if he does not exist.
         */
        private String getImmediateForward(String callee)
        {
                String target = null;

                String query = "Select * from forwards where forwarder = \"" + callee + "\";";
                ResultSet res = dbServer.execute(query);
                try {
                        if (res.first())
                                target = res.getString("forwardTarget");
```

```
                res.close();
        } catch (SQLException e) {
                e.printStackTrace();
        }
        return target;
}
// TODO end

private String getFinalCallee(String callee)
{
        String query = "Select * from forwards where forwarder = \"" + callee + "\";";
        ResultSet res = dbServer.execute(query);
        // loop until the final callee does not forward to any user himself
        try {
                while (res.first()) {
                        callee = res.getString("forwardTarget");
                        query = "select * from forwards where forwarder = \""
                                        + callee + "\";";
                        res.close();
                        res = dbServer.execute(query);
                }
        } catch (SQLException e) {
                e.printStackTrace();
        }
        return callee;
}

public String getFinalCallee(Request request)
{
        Header to     = request.getHeader("To");
        String callee = getName(to);
        return getFinalCallee(callee);
}

private String getName(Header header)
{
        String pattern = "<sip:([^;@]*)[@;]";
        Pattern r = Pattern.compile(pattern);
        Matcher m = r.matcher(header.toString());
        m.find();
        return m.group(1);
}

// TODO kostis 25-3-2015
private boolean checkForCircles(String forwarder, String forwardTarget)
{
        if (forwarder == null || forwardTarget == null)
                return false;

        while (forwardTarget != null) {
```

```
                if (forwarder.equals(forwardTarget))
                        return false;

                forwardTarget = getImmediateForward(forwardTarget);
        }

        return true;
}

// TODO end

// TODO kostis 24-1-2015
private boolean forwardToUser(String forwarder, String forwardTarget)
{
        if (!checkForCircles(forwarder, forwardTarget)) {
                System.out.println("Cycle detected!");
                // TODO kostis 25-3-2015
                unForwardUser(forwarder);
                // TODO end
                return false;
        }

        // delete the previous forwarding entry (if any) for the forwarder
        // of this request
        String query = "Select * from forwards where forwarder = \"" + forwarder + "\";";
        ResultSet res = dbServer.execute(query);
        try {
                if (res.first()) {
                        query = "delete from forwards where forwarder = \""
                                        + forwarder + "\";";
                        if (dbServer.executeUpdate(query) == -1) {
                                System.out.println("Delete Query Failed.");
                                return false;
                        }
                }
        } catch (SQLException e) {
                e.printStackTrace();
                return false;
        }
        // insert the new forwarding entry
        query = "Insert into forwards values(\"" + forwarder + "\", \""
                        + forwardTarget + "\");";
        if (dbServer.executeUpdate(query) == -1) {
                System.out.println("Insert Query Failed.");
                return false;
        }

        return true;
}
```

```
public void handleForwardRequest(RequestEvent requestEvent)
{
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        Header to   = request.getHeader("To");
        String forwarder = null;
        String forwardTarget = null;
        try {
                forwarder = getName(from);
                forwardTarget = getName(to);
        } catch (Exception e) {
                return;
        }

        if (forwardToUser(forwarder, forwardTarget))
                sendForwardTarget(forwardTarget, requestEvent);
        else
                sendForwardTarget("", requestEvent);
}

private void unForwardUser(String forwarder)
{
        String query = "delete from forwards where forwarder = \"" + forwarder + "\"";
        if (dbServer.executeUpdate(query) == -1)
                System.out.println("Delete Query Failed");
}

public void handleUnforwardRequest(RequestEvent requestEvent)
{
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        String forwarder = getName(from);
        unForwardUser(forwarder);
        sendForwardTarget("", requestEvent);
}

public void handleForwardQuestionRequest(RequestEvent requestEvent)
{
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        String forwarder = null;
        forwarder = getName(from);
        String forwardTarget = findForwardTarget(forwarder);
        sendForwardTarget(forwardTarget, requestEvent);
}

private void sendForwardTarget(String forwardTarget, RequestEvent requestEvent)
{
        System.out.println("Sending current forward target: " + forwardTarget);
```

```
            SipProvider sipProvider = (SipProvider) requestEvent.getSource();
            Request request = requestEvent.getRequest();
            HeaderFactory headerFactory = proxy.getHeaderFactory();
            MessageFactory messageFactory = proxy.getMessageFactory();
            Response response;
            try {
                    response = messageFactory.createResponse(Response.OK, request);
            } catch (ParseException e) {
                    ProxyDebug.println("Failed to create response");
                    e.printStackTrace();
                    return;
            }

    // Current Forward Service header
    String headerName  = Constants.SERVICE;
    String headerValue = Constants.GET_CURRENT_FORWARD;
    Header serviceHeader;
            try {
                    serviceHeader = headerFactory.createHeader(headerName, headerValue);
            } catch (ParseException e) {
        ProxyDebug.println("An unexpected error occurred while constructing the Service Header");
        e.printStackTrace();
        return;
            }
            response.addHeader(serviceHeader);

    // Current forwarding target header
    String currentTarget = Constants.CURRENT_TARGET;
    Header currentTargetHeader;
            try {
                    currentTargetHeader = headerFactory.createHeader(currentTarget, forwardTarget);
            } catch (ParseException e) {
        ProxyDebug.println("An unexpected error occurred while constructing the Service Header");
        e.printStackTrace();
        return;
            }
            response.addHeader(currentTargetHeader);

            ServerTransaction serverTransaction = requestEvent.getServerTransaction();
            try {
                    if (serverTransaction!=null)
                            serverTransaction.sendResponse(response);
                    else
                            sipProvider.sendResponse(response);
            } catch (SipException e) {
                    ProxyDebug.println("Failed to send current target response");
                    e.printStackTrace();
            }
    }
```

```
        private String findForwardTarget(String forwarder)
        {
                String query = "Select * from forwards where forwarder = \""
                                + forwarder + "\";";
                ResultSet res = dbServer.execute(query);
                String forwardTarget = "";
                try {
                        if (res.first()) {
                                forwardTarget = res.getString("forwardTarget");
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return forwardTarget;
        }
        // TODO end
}
```

Η κλάση αυτή υλοποιεί την επικοινωνία με τη βάση δεδομένων. Όταν κάποιος χρήστης, A, καλέσει τον χρήστη B που προωθεί σε κάποιον άλλον χρήστη, τότε θα πρέπει να αναζητηθεί η αλυσίδα προωθήσεων και να μην ευρεθούν κύκλοι. Για το λόγο αυτό, αναζητούμε κύκλους προωθήσεων και αν δεν ευρεθούν, τελικώς προωθούμε την κλήση στον τελικό παραλήπτη της. Για να ελέγξουμε ότι δεν υπάρχουν κύκλοι, ελέγχουμε αν σε κάποιο σημείο της αλυσίδας ο forwarder συμπίπτει με τον forwardTarget.

4.2.4 Billing

BillingFrame.java

```
package net.java.sip.communicator.gui.softeng;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
```

```
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;

import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;
import net.java.sip.communicator.sip.softeng.BillingProcessing;
import net.java.sip.communicator.sip.softeng.Constants;

// TODO kostis 10-4-2015
public class BillingFrame extends JDialog
{
    private static final long serialVersionUID = 1L;
    private JLabel billLabel;
    private JLabel billStrategy;
    private JComboBox<String> billStrategies;
    private BillingProcessing billingProcessing;
    private final String CMD_CHANGE = "cmd.change";

    /**
     * Creates a new Billing Settings Dialog
     * @param sipManager
     * @param sipManager
     */
    public BillingFrame(Frame parent, SipManager sipManager, boolean modal)
    {
        super(parent, modal);
        this.billingProcessing = sipManager.getBillingProcessing();
        initComponents();
        pack();
        centerWindow();
    }

    /**
     * Centers the window on the screen.
     */
    private void centerWindow()
    {
        Rectangle screen = new Rectangle(
                Toolkit.getDefaultToolkit().getScreenSize());
        Point center = new Point(
                (int) screen.getCenterX(), (int) screen.getCenterY());
        Point newLocation = new Point(
                center.x - this.getWidth() / 2, center.y - this.getHeight() / 2);
        if (screen.contains(newLocation.x, newLocation.y,
                this.getWidth(), this.getHeight())) {
            this.setLocation(newLocation);
        }
    }
```

```
/**
 * Creates the contents of the Billing Settings Dialog
 */
private void initComponents()
{
    Container contents = getContentPane();
    contents.setLayout(new BorderLayout());

    setTitle("Billing Settings");
    getAccessibleContext().setAccessibleDescription("Billing Settings");
     JPanel centerPane = new JPanel();
    centerPane.setLayout(new GridBagLayout());
    int gridy = 1;

    // Current billing strategy title
    JLabel strategyLabel = new JLabel("Current Strategy:");
    GridBagConstraints c = new GridBagConstraints();
    c.gridx     = 0;
    c.gridy     = gridy;
    c.anchor    = GridBagConstraints.WEST;
    c.fill      = GridBagConstraints.HORIZONTAL;
    c.weighty   = 0;
    c.insets    = new Insets(12,12,0,12);
    centerPane.add(strategyLabel, c);

    // The name of the current billing strategy
    billStrategy = new JLabel(Constants.PER_SECOND);
    c = new GridBagConstraints();
    c.gridx     = 1;
    c.gridy     = gridy++;
    c.anchor    = GridBagConstraints.WEST;
    c.fill      = GridBagConstraints.HORIZONTAL;
    c.weighty   = 0;
    c.insets    = new Insets(12,0,0,12);
    centerPane.add(billStrategy, c);

    // Drop down menu to select the billing package
    String[] strategies = {Constants.PER_SECOND, Constants.FIXED, Constants.FREE};
    billStrategies = new JComboBox<String>(strategies);
    billStrategies.setPreferredSize(new Dimension(120, 25));
    c = new GridBagConstraints();
    c.gridx   = 0;
    c.gridy   = gridy;
    c.anchor  = GridBagConstraints.WEST;
    c.fill    = GridBagConstraints.HORIZONTAL;
    c.weighty = 0;
    c.insets=new Insets(12,12,0,12);
    centerPane.add(billStrategies, c);

    // Change strategy button
```

```
        JButton changeButton = new JButton();
        changeButton.setText("Change");
        changeButton.setPreferredSize(new Dimension(120, 25));
        changeButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                clickHandler(CMD_CHANGE);
            }

        });
        c = new GridBagConstraints();
        c.gridx   = 1;
        c.gridy   = gridy++;
        c.anchor  = GridBagConstraints.WEST;
        c.fill    = GridBagConstraints.HORIZONTAL;
        c.weightx = 1;
        c.weighty = 0;
        c.insets  = new Insets(12,0,0,12);
        centerPane.add(changeButton, c);

        // label showing the current bill
        billLabel = new JLabel(" Retreiving Data...");
        c = new GridBagConstraints();
        c.gridx   = 0;
        c.gridy   = gridy++;
        c.anchor  = GridBagConstraints.WEST;
        c.weighty = 0;
        c.insets  = new Insets(12,12,12,12);
        centerPane.add(billLabel, c);

        contents.add(centerPane, BorderLayout.CENTER);
    }

    /**
     * Handles a click event
     */
    private void clickHandler(String cmd)
    {
        if (cmd.equals(CMD_CHANGE)) {
            String strategy = (String) billStrategies.getSelectedItem();
            try {
                billingProcessing.setStrategy(strategy);
            } catch (CommunicationsException e) {
                e.printStackTrace();
            }
        }
    }

    /**
```

```
     * Sets the total bill label
     * @param bill the new total bill
     */
    public void setTotalBill(Long bill)
    {
        billLabel.setText("Total Bill :   " + bill + " euros.");
    }

    public void setCurrentStrategy(String strategy)
    {
        billStrategy.setText(strategy);
    }
    // TODO kostis
}
```

BillingProcessing.java

```
package net.java.sip.communicator.sip.softeng;

import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;

public class BillingProcessing
{
    private TotalBillRequestSender totalBillRequestSender;
    private StrategySender byRateStrategySender;
    private StrategySender fixedStrategySender;
    private StrategySender freeStrategySender;
    private GetStrategySender getStrategySender;

    public BillingProcessing(SipManager sipManCallback)
    {
        totalBillRequestSender = new TotalBillRequestSender(sipManCallback);
        byRateStrategySender   = new StrategySender(sipManCallback, Constants.CHARGE_BY_RATE);
        fixedStrategySender    = new StrategySender(sipManCallback, Constants.CHARGE_FIXED_PRICE);
        freeStrategySender     = new StrategySender(sipManCallback, Constants.FREE_OF_CHARGE);
        getStrategySender      = new GetStrategySender(sipManCallback);
    }

    public void requestTotalBill() throws CommunicationsException
    {
        totalBillRequestSender.sendRequest("dummy"); // dummy target
    }

    public void setStrategy(String strategy) throws CommunicationsException
    {
        System.out.println("Sending request to change strategy to: " + strategy);

        if (strategy.equals(Constants.CHARGE_BY_RATE))
```

```
            byRateStrategySender.sendRequest("dummy");
        else if (strategy.equals(Constants.CHARGE_FIXED_PRICE))
            fixedStrategySender.sendRequest("dummy");
        else if (strategy.equals(Constants.FREE_OF_CHARGE))
            freeStrategySender.sendRequest("dummy");
    }

    public void getStrategy() throws CommunicationsException
    {
        getStrategySender.sendRequest("dummy");
    }
}
```

Η κλάση BillingProcessing είναι η κλάση που αναλαμβάνει να χειριστεί τα μηνύματα που έρχονται από τον χρήστη. Έτσι, ανάλογα με το αν ο χρήστης έχει ζητήσει να υπολογιστεί ο τελικός του λογαριασμός, να αλλάξει ή να δει τη στρατηγική του, η BillingProcessing αναλαμβάνει να προωθήσει το αίτημα στις κατάλληλες υποκλάσεις.

GetStrategySender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class GetStrategySender extends RequestSender
{
    public GetStrategySender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected void addCustomHeaders(Request request) throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.CURRENT_STRATEGY;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
    }
}
```

FixedStrategySender.java

```java
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;

import net.java.sip.communicator.sip.SipManager;

public class FixedStrategySender extends RequestSender
{
    public FixedStrategySender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected Header createCustomHeader() throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.CHARGE_FIXED_PRICE;
        return sipManCallback.headerFactory.createHeader(headerName, headerValue);
    }
}
```

FreeStrategySender.java

```java
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;

import net.java.sip.communicator.sip.SipManager;

public class FreeStrategySender extends RequestSender
{
    public FreeStrategySender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected Header createCustomHeader() throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.FREE_OF_CHARGE;
        return sipManCallback.headerFactory.createHeader(headerName, headerValue);
```

```
    }
}
```

PerSecondStrategySender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;

import net.java.sip.communicator.sip.SipManager;

public class PerSecondStrategySender extends RequestSender
{
    public PerSecondStrategySender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected Header createCustomHeader() throws ParseException
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.CHARGE_BY_RATE;
        return sipManCallback.headerFactory.createHeader(headerName, headerValue);
    }
}
```

TotalBillRequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;

import javax.sip.header.Header;
import javax.sip.message.Request;

import net.java.sip.communicator.sip.SipManager;

public class TotalBillRequestSender extends RequestSender
{
    public TotalBillRequestSender(SipManager sipManCallback)
    {
        super(sipManCallback);
    }

    @Override
    protected void addCustomHeaders(Request request) throws ParseException
```

```
    {
        String headerName = Constants.SERVICE;
        String headerValue = Constants.GET_BILL;
        Header header = sipManCallback.headerFactory.createHeader(headerName, headerValue);
        request.addHeader(header);
    }
}
```

Κάθε μία από τις παραπάνω κλάσεις, φτιάχνει κατάλληλα το Header έτσι ώστε να προωθηθεί στον server το αίτημα του χρήστη.

BillingFactory.java

```
package gov.nist.sip.proxy.softeng;

public class BillingFactory
{
    public static CostCalculator createCostCalculator(String strategy) {
        CostCalculator costCalculator = null;
        switch(strategy) {
        case Constants.CHARGE_FIXED_PRICE:
            costCalculator = new ChargeFixedPriceCalculator();
            break;
        case Constants.CHARGE_BY_RATE:
            costCalculator = new ChargeByRateCalculator();
            break;
        case Constants.FREE_OF_CHARGE:
            costCalculator = new FreeOfChargeCalculator();
            break;
        default:
            costCalculator = new ChargeByRateCalculator();
            break;
        }
        return costCalculator;
    }
}
```

Η κλάση αυτή, ανάλογα με το τι είδους στρατηγική έχει επιλεγεί, επιστρέφει ένα κατάλληλο αντικείμενο, το οποίο θα υπολογίσει το κόστος σύμφωνα με την επιλεγμένη στρατηγική (Factory Design Pattern).

BillingServer.java

```
package gov.nist.sip.proxy.softeng;

import gov.nist.sip.proxy.Proxy;
import gov.nist.sip.proxy.ProxyDebug;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.sip.RequestEvent;
import javax.sip.ServerTransaction;
import javax.sip.SipException;
import javax.sip.SipProvider;
import javax.sip.header.CallIdHeader;
import javax.sip.header.Header;
import javax.sip.header.HeaderFactory;
import javax.sip.message.MessageFactory;
import javax.sip.message.Request;
import javax.sip.message.Response;

public class BillingServer {

    protected class Call {
        private String caller;
        private String callId;
        private String callee;
        private String callerBillingStrategy;
        private RequestEvent callerRequestEvent;
        private CostCalculator costCalculator;

        public Call(String caller, String callId, String callee, RequestEvent requestEvent,
                            String callerBillingStrategy) {
            this.caller     = caller;
            this.callId     = callId;
            this.callee     = callee;
            this.callerRequestEvent = requestEvent;
            this.callerBillingStrategy = callerBillingStrategy;
            this.costCalculator = BillingFactory.createCostCalculator(this.callerBillingStrategy);
        }

        public String getCallId() {
            return this.callId;
        }

        public String getCaller() {
            return this.caller;
        }

        public String getCallee() {
            return this.callee;
        }
```

```
        public void setTimeStart(long timeStart) {
            this.costCalculator.setTimeStart(timeStart);
        }

        public void printCall() {
            System.out.println("CallId : " + this.callId + ", Caller : " + this.caller +
                               ", Callee : " + this.callee);
        }

        public RequestEvent getCallerRequestEvent() {
            return this.callerRequestEvent;
        }

        public String getCallerBillingStrategy() {
            return this.callerBillingStrategy;
        }

        public void setTimeEnd(long timeEnd) {
            this.costCalculator.setTimeEnd(timeEnd);
        }

        public long calculateCost() {
            return this.costCalculator.calculateCost();
        }
    }

    private DBServer dbServer;
    private Proxy proxy;
    private ArrayList<Call> calls = null;
    private static String defaultStrategy;

    private static BillingServer pInstance;

    private BillingServer()
    {
        defaultStrategy = Constants.CHARGE_BY_RATE;
        calls           = new ArrayList<Call>();
    }

    public static BillingServer getInstance()
    {
        if (pInstance == null)
            pInstance = new BillingServer();

        return pInstance;
    }

    public void init(DBServer dbServer, Proxy proxy)
    {
        this.dbServer = dbServer;
```

```
        this.proxy    = proxy;
    }

    public void callCreated(RequestEvent requestEvent)
    {
        String caller = getName(requestEvent.getRequest().getHeader("From"));
        String callId = ((CallIdHeader) requestEvent.getRequest().getHeader(
                        CallIdHeader.NAME)).getCallId();
        String callee = getName(requestEvent.getRequest().getHeader("To"));
        String callerBillingStrategy = getCurrentStrategy(caller);
        System.out.println("Call created. Caller: " + caller
                + " has strategy: " + callerBillingStrategy);
        Call call = new Call(caller, callId, callee, requestEvent, callerBillingStrategy);
        calls.add(call);
        System.out.println("A new call was added");
        System.out.println("These are the active calls:");
        printCalls();
    }

    private void printCalls() {
        for (Call call: calls) {
            call.printCall();
        }
    }

    public void callStarted(RequestEvent requestEvent) {
        String callId = ((CallIdHeader)
                requestEvent.getRequest().getHeader(CallIdHeader.NAME)).getCallId();
        long timeStart = System.currentTimeMillis();
        for (Call call: calls) {
            if (call.getCallId().equals(callId)) {
                call.setTimeStart(timeStart);
                break;
            }
        }
    }

    public void callEnded(RequestEvent requestEvent) {
        String callId = ((CallIdHeader)
            requestEvent.getRequest().getHeader(CallIdHeader.NAME)).getCallId();
        long timeEnd = System.currentTimeMillis();
        String caller = "";
        Call callToRemove = null;
        for (Call call: calls) {
            if (call.getCallId().equals(callId)) {
                caller = call.getCaller();
                callToRemove = call;
                break;
            }
        }
```

```
        if (callToRemove == null) {
            System.out.println("No call with that callId exists.");
            return;
        }
        callToRemove.setTimeEnd(timeEnd);
        long bill = callToRemove.calculateCost();
        updateTotalBill(caller, bill);
        calls.remove(callToRemove);
    }

    private String getName(Header header)
    {
        String pattern = "<sip:([^;@]*)[@;]";
        Pattern r = Pattern.compile(pattern);
        Matcher m = r.matcher(header.toString());
        m.find();
        return m.group(1);
    }

    public void handleShowBillRequest(RequestEvent requestEvent)
    {
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        String billedUser = null;
        try {
            billedUser = getName(from);
        } catch (Exception e) {
            ProxyDebug.println("Could not get the user name from the request");
            return;
        }
        long totalBill = getTotalBill(billedUser);
        sendBillToUser(billedUser, requestEvent, totalBill);
    }

    private long getTotalBill(String billedUser)
    {
        String query = "Select * from bills where user = \"" + billedUser + "\";";
        ResultSet res = dbServer.execute(query);
        long billTotal = 0;
        try {
            if (res.first()) {
                billTotal = res.getLong("totalBill");
            }
        } catch (Exception e) {
            return 0L;
        }
        return billTotal;
    }

    private void updateTotalBill(String user, long bill) {
```

```
        String query = "Select * from bills where user = \"" + user + "\";";
        ResultSet res = dbServer.execute(query);
        try {
            if (res.first()) {
                query = "Update bills set totalBill = totalBill + \""
                        + bill + "\" where user = \"" + user + "\";";
                if (dbServer.executeUpdate(query) == -1)
                    System.out.println("Update Query on bills Failed");
            }
            else {
                query = "Insert into bills values(\"" + user +
                           "\", \"" + bill + "\");";
                if (dbServer.executeUpdate(query) == -1)
                System.out.println("Insert Query Failed");
            }
        } catch (Exception e) {
            return;
        }
    }

    private void sendBillToUser(String billedUser, RequestEvent requestEvent, long bill)
    {
        System.out.println("Sending bill of " + bill + " euros to user: " + billedUser);

        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        Request request = requestEvent.getRequest();
        HeaderFactory headerFactory = proxy.getHeaderFactory();
        MessageFactory messageFactory = proxy.getMessageFactory();
        Response response;
        try {
            response = messageFactory.createResponse(Response.OK, request);
        } catch (ParseException e) {
            ProxyDebug.println("Failed to create response");
            e.printStackTrace();
            return;
        }

        // Get Bill Service header
        String headerName  = Constants.SERVICE;
        String headerValue = Constants.GET_BILL;
        Header serviceHeader;
        try {
            serviceHeader = headerFactory.createHeader(headerName, headerValue);
        } catch (ParseException e) {
            ProxyDebug.println("An unexpected error occurred while constructing the Service Header");
            e.printStackTrace();
            return;
        }
        response.addHeader(serviceHeader);
```

```
        // Get Bill Content header
        String billHeaderName = Constants.BILL;
        String billString = Long.toString(bill);
        Header billHeader;
        try {
            billHeader = headerFactory.createHeader(billHeaderName, billString);
        } catch (ParseException e) {
            ProxyDebug.println("An unexpected error occurred while constructing the Bill Content Header");
            e.printStackTrace();
            return;
        }
        response.addHeader(billHeader);

        ServerTransaction serverTransaction = requestEvent.getServerTransaction();
        try {
            if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else
                sipProvider.sendResponse(response);
        } catch (SipException e) {
            ProxyDebug.println("Failed to send Bill response");
            e.printStackTrace();
        }
    }

    /**
     * Handles a request to change the current billing strategy
     * @param requestEvent
     */
    public void handleChangeStrategyRequest(RequestEvent requestEvent)
    {
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        String username = null;
        try {
            username = getName(from);
        } catch (Exception e) {
            return;
        }

        Header billHeader = request.getHeader(Constants.SERVICE);
        String strategy = billHeader.toString().trim().replace(Constants.SERVICE + ": ", "");

        changeStrategy(username, strategy);
        sendStrategy(requestEvent, strategy);
    }

    /**
     * Changes the strategy for that user to the given strategy.
     * @param username
```

```
     * @param newStrategy
     */
    private void changeStrategy(String username, String newStrategy)
    {
        removeStrategy(username);
        setStrategy(username, newStrategy);
    }

    /**
     * Returns a response to the user informing him that this is his current billing strategy
     * @param requestEvent
     * @param strategy
     */
    private void sendStrategy(RequestEvent requestEvent, String strategy)
    {
        System.out.println("Sending current strategy: " + strategy);

        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        Request request = requestEvent.getRequest();
        HeaderFactory headerFactory = proxy.getHeaderFactory();
        MessageFactory messageFactory = proxy.getMessageFactory();
        Response response;
        try {
            response = messageFactory.createResponse(Response.OK, request);
        } catch (ParseException e) {
            ProxyDebug.println("Failed to create response");
            e.printStackTrace();
            return;
        }

        // Current Strategy Service header
        String headerName  = Constants.SERVICE;
        String headerValue = Constants.CURRENT_STRATEGY;
        Header serviceHeader;
        try {
            serviceHeader = headerFactory.createHeader(headerName, headerValue);
        } catch (ParseException e) {
            ProxyDebug.println("An unexpected error occurred while constructing the Service Header");
            e.printStackTrace();
            return;
        }
        response.addHeader(serviceHeader);

        // Current forwarding target header
        String currentStrategy = Constants.CURRENT_STRATEGY;
        Header currentStrategyHeader;
        try {
            currentStrategyHeader = headerFactory.createHeader(currentStrategy, strategy);
        } catch (ParseException e) {
            ProxyDebug.println("An unexpected error occurred while constructing a Message Header");
```

```
            e.printStackTrace();
            return;
        }
        response.addHeader(currentStrategyHeader);

        ServerTransaction serverTransaction = requestEvent.getServerTransaction();
        try {
            if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else
                sipProvider.sendResponse(response);
        } catch (SipException e) {
            ProxyDebug.println("Failed to send current target response");
            e.printStackTrace();
        }
    }

    /**
     * Removes the strategy for that user if it exists
     * @param username
     */
    private boolean removeStrategy(String username)
    {
        String query = "Delete from BillingStrategies where username = \"" + username + "\";";
        if (dbServer.executeUpdate(query) == -1) {
            System.out.println("Delete Query Failed for BillingStrategies");
            return false;
        }
        return true;
    }

    /**
     * Sets the strategy for the given user.
     * Assumes no strategy exists for that user.
     * @param username
     * @param newStrategy
     */
    private boolean setStrategy(String username, String newStrategy)
    {
        String query = "Insert into BillingStrategies values(\"" +
                        username + "\", \"" + newStrategy + "\");";

        if (dbServer.executeUpdate(query) == -1) {
            System.out.println("Setting Billing Strategy Failed.");
            return false;
        }
        return true;
    }

    /**
```

```
     * Returns the current billing strategy for the user.
     * Inserts the default strategy to the database and returns it if none exists.
     * @param username
     */
    public String getCurrentStrategy(String username)
    {
        String query = "Select * from BillingStrategies where username = \"" + username + "\"";
        ResultSet res = dbServer.execute(query);
        String strategy = null;
        try {
            if (res.first())
                strategy = res.getString("strategy");
        } catch (SQLException e) {
            e.printStackTrace();
        }

        if (strategy == null) { // no strategy exists
            setStrategy(username, defaultStrategy);
            return defaultStrategy;
        }

        return strategy;
    }

    /**
     * Handles a request to get the current billing strategy.
     * @param requestEvent
     */
    public void handleCurrentStrategyRequest(RequestEvent requestEvent)
    {
        Request request = requestEvent.getRequest();
        Header from = request.getHeader("From");
        String username = null;
        try {
            username = getName(from);
        } catch (Exception e) {
            return;
        }

        String strategy = getCurrentStrategy(username);
        sendStrategy(requestEvent, strategy);
    }

    public static String getDefaultStrategy()
    {
        return defaultStrategy;
    }
}
```

Σε αυτή την κλάση, έχουμε μεθόδους σχετικά με την αρχή μιας κλήσης, με το τέλος καθώς και την ενημέρωση των χρηστών για τον λογαριασμό. Επιπλέον, γίνεται ο χειρισμός των στρατηγικών χρέωσης. Περιληπτικά οι μέθοδοί μας είναι εξής:

- `callCreated`: γίνεται establish μία νέα κλήση, δίνουμε `callID`, καθορίζουμε τον caller και τον callee και επίσης, βρίσκουμε την στρατηγική χρέωσης του caller.
- `printCalls`: εκτυπώνονται οι κλήσεις
- `callStarted`: βρίσκουμε τον χρόνο (από τα milliseconds του συστήματος) που άρχισε η κλήση και με βάση το callID της, την αποθηκεύουμε
- `callEnded`: βρίσκουμε τη στιγμή που τελειώνει η κλήση, αποθηκεύουμε το χρόνο λήξης και στη συνέχεια υπολογίζουμε το κόστος ανάλογα με την στρατηγική του χρήστη. Επιπλέον, μετά τον υπολογισμό του κόστους και το τέλος της κλήσης, διαγράφω την κλήση από τις ενεργές.
- `handleShowBillRequest`: βρίσκουμε τον λογαριασμό του χρήστη και τα στέλνουμε στον χρήστη που έκανε την κλήση
- `getTotablBill`: κάνουμε query στη βάση και παίρνουμε το λογαριασμό
- `sendBillToUser`: δημιουργούμε ένα header και το στέλνουμε
- `handleChangeStrategyRequest`: χειριζόμαστε μία αλλαγή τιμολογιακής στρατηγικής
- `changeStrategy`: αλλάζουμε τη στρατηγική ενός χρήστη με τη στρατηγική που δόθηκε
- `sendStrategy`: στέλνουμε response στον χρήστη σχετικά με την τρέχουσα τιμολογιακή πολιτική
- `removeStrategy`: αφαιρεί τη στρατηγική του χρήστη, αν υπάρχει στρατηγική
- `setStrategy`: Καθορίζει μία στρατηγική για τον δεδομένο χρήστη. Υποθέτουμε ότι δεν υπάρχει άλλη στρατηγική για αυτό το χρήστη.
- `getCurrentStrategy`: Επιστρέφει την τρέχουσα τιμολογιακή στρατηγική. Αν δεν υπάρχει, τότε επιστρέφει την default τιμολογιακή πολιτική.

- `handleCurrentStrategyRequest`: Χειρίζεται ένα request για την τρέχουσα τιμολογιακή πολιτική.

ChargeByRateCalculator.java

```
package gov.nist.sip.proxy.softeng;

public class ChargeByRateCalculator extends CostCalculator {
    private long rate = Constants.RATE;

    @Override
    public long calculateCost() {
        System.out.println("Calc ByRate");
        System.out.println("Start = " + timeStart);
        System.out.println("End = " + timeEnd);
        long callTime = (this.timeEnd - this.timeStart)/1000;
        return rate * callTime;
    }
}
```

Για να υπολογίσουμε το κόστος με αυτή τη στρατηγική, έχουμε ένα σταθερό ποσοστό χρέωσης και πολλαπλασιάζουμε με αυτό τη διάρκεια κλήσης.

ChargeFixedPriceCalculator.java

```
package gov.nist.sip.proxy.softeng;

public class ChargeFixedPriceCalculator extends CostCalculator {
    private long fixedCost = Constants.FIXED_PRICE;

    @Override
    public long calculateCost() {
        System.out.println("Calc Fixed");
        return fixedCost;
    }

}
```

Εδώ το κόστος είναι σταθερό, οπότε για τον υπολογισμό της χρέωσης δεν ασχολούμαστε καν με τη διάκρεια κλήσης.

FreeOfChargeCalculator.java

```
package gov.nist.sip.proxy.softeng;

public class FreeOfChargeCalculator extends CostCalculator {

    @Override
```

```
    public long calculateCost() {
        System.out.println("Calc Free");
        return 0;
    }

}
```

Εδώ το κόστος κλήσης είναι μηδενικό ανεξάρτητα και πάλι από τη διάρκεια.

CostCalculator.java

```
package gov.nist.sip.proxy.softeng;

public abstract class CostCalculator {
    protected long timeStart;
    protected long timeEnd;

    public long getTimeStart() {
        return timeStart;
    }

    public void setTimeStart(long timeStart) {
        this.timeStart = timeStart;
    }

    public long getTimeEnd() {
        return timeEnd;
    }

    public void setTimeEnd(long timeEnd) {
        this.timeEnd = timeEnd;
    }

    public abstract long calculateCost();

}
```

RequestSender.java

```
package net.java.sip.communicator.sip.softeng;

import java.text.ParseException;
import java.util.ArrayList;

import javax.sip.ClientTransaction;
import javax.sip.InvalidArgumentException;
import javax.sip.SipException;
import javax.sip.TransactionUnavailableException;
import javax.sip.address.Address;
```

```
import javax.sip.address.URI;
import javax.sip.header.CSeqHeader;
import javax.sip.header.CallIdHeader;
import javax.sip.header.FromHeader;
import javax.sip.header.Header;
import javax.sip.header.MaxForwardsHeader;
import javax.sip.header.ToHeader;
import javax.sip.message.Request;

import net.java.sip.communicator.common.Console;
import net.java.sip.communicator.common.PropertiesDepot;
import net.java.sip.communicator.common.Utils;
import net.java.sip.communicator.sip.CallProcessing;
import net.java.sip.communicator.sip.CommunicationsException;
import net.java.sip.communicator.sip.SipManager;

public abstract class RequestSender
{
    protected static final Console console = Console.getConsole(CallProcessing.class);
    protected SipManager sipManCallback;

    public RequestSender(SipManager sipManCallback)
    {
        this.sipManCallback = sipManCallback;
    }

    protected abstract void addCustomHeaders(Request request) throws ParseException;

    /*
     * Template Method used to send INFO messages with custom headers
     */
    public void sendRequest(String target) throws CommunicationsException
    {
        try
        {
            console.logEntry();

            target = target.trim();
            // Remove excessive characters from phone numbers such as ' ','(',')','-'
            String excessiveChars = Utils.getProperty(
                    "net.java.sip.communicator.sip.EXCESSIVE_URI_CHARACTERS");

            //--------------------------------------------------------------------------
            // an ugly hack to override old xml configurations (todo: remove at some point)
            // define excessive chars for sipphone.com users
            String isSipphone = Utils.getProperty(
                    "net.java.sip.communicator.sipphone.IS_RUNNING_SIPPHONE");
            if(excessiveChars == null && isSipphone != null && isSipphone.equalsIgnoreCase("true"))
            {
                excessiveChars = "( )-";
```

```
            PropertiesDepot.setProperty(
                 "net.java.sip.communicator.sip.EXCESSIVE_URI_CHARACTERS", excessiveChars);
            PropertiesDepot.storeProperties();
        }
        // -----------------------------------------------------------------------

        if(excessiveChars != null )
        {
            StringBuffer calleeBuff = new StringBuffer(target);
            for(int i = 0; i < excessiveChars.length(); i++)
            {
                String charToDeleteStr = excessiveChars.substring(i, i+1);

                int charIndex = -1;
                while( (charIndex = calleeBuff.indexOf(charToDeleteStr)) != -1)
                    calleeBuff.delete(charIndex, charIndex + 1);
            }
            target = calleeBuff.toString();
        }

        // Handle default domain name (i.e. transform 1234 -> 1234@sip.com
        String defaultDomainName =
            Utils.getProperty("net.java.sip.communicator.sip.DEFAULT_DOMAIN_NAME");
        if (defaultDomainName != null //no sip scheme
            && !target.trim().startsWith("tel:")
            && target.indexOf('@') == -1 //most probably a sip uri
            ) {
            target = target + "@" + defaultDomainName;
        }

        // Let's be uri fault tolerant
        if (target.toLowerCase().indexOf("sip:") == -1 //no sip scheme
            && target.indexOf('@') != -1 //most probably a sip uri
            ) {
            target = "sip:" + target;

        }

        // Request URI
        URI requestURI;
        try {
            requestURI = sipManCallback.addressFactory.createURI(target);
        }
        catch (ParseException ex) {
            console.error(target + " is not a legal SIP uri!", ex);
            throw new CommunicationsException(target + " is not a legal SIP uri!", ex);
        }

        // Call ID
        CallIdHeader callIdHeader = sipManCallback.sipProvider.getNewCallId();
```

```
// CSeq
CSeqHeader cSeqHeader;
try {
    cSeqHeader = sipManCallback.headerFactory.createCSeqHeader(1, Request.INFO);
}
catch (ParseException ex) { //Shouldn't happen
    console.error(ex, ex);
    throw new CommunicationsException(
        "An unexpected error occurred while"
        + "constructing the CSeqHeadder", ex);
}
catch (InvalidArgumentException ex) { //Shouldn't happen
    console.error(
        "An unexpected error occurred while"
        + "constructing the CSeqHeadder", ex);
    throw new CommunicationsException(
        "An unexpected error occurred while"
        + "constructing the CSeqHeadder", ex);
}

// FromHeader
FromHeader fromHeader = sipManCallback.getFromHeader();
// ToHeader
Address toAddress = sipManCallback.addressFactory.createAddress(requestURI);
ToHeader toHeader;
try {
    toHeader = sipManCallback.headerFactory.createToHeader(toAddress, null);
}
catch (ParseException ex) { //Shouldn't happen
    console.error(
        "Null is not an allowed tag for the to header!", ex);
    throw new CommunicationsException(
        "Null is not an allowed tag for the to header!", ex);
}

// ViaHeaders
@SuppressWarnings("rawtypes")
ArrayList viaHeaders = sipManCallback.getLocalViaHeaders();
// MaxForwards
MaxForwardsHeader maxForwards = sipManCallback.getMaxForwardsHeader();


Request request = null;
try {
    request = sipManCallback.messageFactory.createRequest(requestURI,
        Request.INFO, callIdHeader, cSeqHeader, fromHeader,
        toHeader, viaHeaders, maxForwards);
}
catch (ParseException ex) {
    console.error(
```

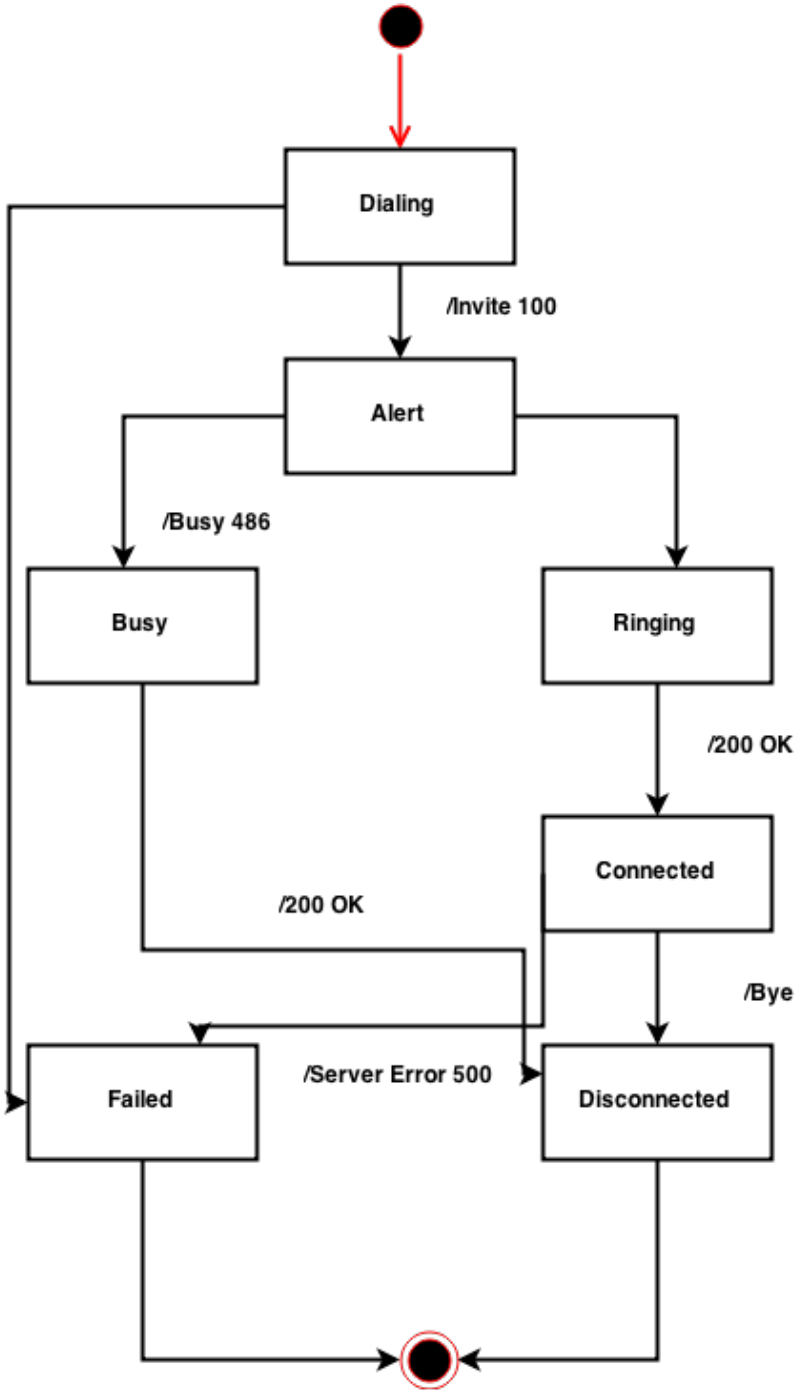
```
                    "Failed to create Request!", ex);
                throw new CommunicationsException(
                    "Failed to create Request!", ex);
            }

            // Add the custom headers for each service
            try {
                addCustomHeaders(request);
            } catch (ParseException e) {
                console.error("An unexpected error occurred while constructing a header", e);
                throw new CommunicationsException(
                        "An unexpected error occurred while constructing a header", e);
            }

            // Transaction
            ClientTransaction transaction;
            try {
                transaction = sipManCallback.sipProvider.
                    getNewClientTransaction(request);
            }
            catch (TransactionUnavailableException ex) {
                console.error(
                    "Failed to create transaction.\n" +
                    "This is most probably a network connection error.", ex);
                throw new CommunicationsException(
                    "Failed to create transaction.\n" +
                    "This is most probably a network connection error.", ex);
            }
            try {
                transaction.sendRequest();
                if( console.isDebugEnabled() )
                    console.debug("sent request: " + request);
            }
            catch (SipException ex) {
                console.error(
                    "An error occurred while sending a request", ex);
                throw new CommunicationsException(
                    "An error occurred while sending a request", ex);
            }
        }
        finally
        {
            console.logExit();
        }
    }
}
```

5 Διαγράμματα Κατάστασης

Παραθέτουμε παρακάτω το διάγραμμα κατάστασης για μία κλήση:



Σχήμα 22: Διάγραμμα κατάστασης για κανονική κλήση

6 Ανοιχτά Ζητήματα

Ως ανοιχτό ζήτημα αφήνουμε το ζήτημα της ασφάλειας. Θεωρούμε ότι αποθηκεύοντας τους κωδικούς των χρηστών, δεν είμαστε αρκετά καλυμμένοι για την περίπτωση υποκλοπής. Θα μπορούσε, λοιπόν, να αναπτυχθεί κάποιο πρωτόκολλο επικοινωνίας που περιλαμβάνει κρυπτογραφημένες πληροφορίες για τα credentials των χρηστών.

Ένα ακόμη ζήτημα που θα ήταν ενδιαφέρον να αναπτυχθεί στο μέλλον αφορά στην ουσία της επικοινωνίας των χρηστών και συγκεκριμένα στη δυνατότητα ηχογραφημένων μηνυμάτων. Θα μπορούσε λόγου χάρη, ένας χρήστης A, όταν ο χρήστης B δεν απαντά να του στέλνει ένα ηχογραφημένο μήνυμα, σαν να είχε ο χρήστης B τηλεφωνητή.

Τέλος, θα ήταν ουσιαστικό και πιο ενδιαφέρον να επεκταθεί η τιμολογιακή πολιτική των κλήσεων και να στραφεί σε πιο ρεαλιστικές μεθόδους τιμολόγησης.

7 Λεξικό

7.1 Ορολογία και Συντομογραφίες

Όλοι οι όροι που χρησιμοποιήθηκαν βρίσκονται επεξηγημένοι στο κείμενο. Συντομογραφίες δεν υπάρχουν.