added: sip-proxy/src/gov/nist/sip/proxy/Chargement.java

```
@ Chargement.java:3 @
package gov.nist.sip.proxy;

public class Chargement {
        public static Double normalChargement = 0.03;
        public static Double premiumChargement = 0.01;
}
```

modified: sip-proxy/src/gov/nist/sip/proxy/Proxy.java

```
@ Proxy.java:25 @ import gov.nist.javax.sip.header.*;
/*
import sim.java.net.*;
//endif
*/
 */


/** Proxy Entry point.
@ Proxy.java:39 @ import sim.java.net.*;
 *
 */
public class Proxy implements SipListener  {

    protected LinkedList listeningPoints;
    // Map the server transactions with the client transactions
    protected SipStack sipStack;
    protected SipProvider defaultProvider;

    protected MessageFactory messageFactory;
    protected HeaderFactory headerFactory;
    protected AddressFactory addressFactory;

    protected Configuration configuration;
    protected PresenceServer presenceServer;

    protected Registrar registrar;
    protected ProxyUtilities proxyUtilities;
    protected Authentication authentication;
    protected RequestForwarding requestForwarding;
    protected ResponseForwarding responseForwarding;


    public RequestForwarding getRequestForwarding() {
        return requestForwarding;
    }

    public ResponseForwarding getResponseForwarding() {
        return responseForwarding;
    }

    public AddressFactory getAddressFactory() {
        return addressFactory;
    }

    public MessageFactory getMessageFactory() {
        return messageFactory;
    }

    public HeaderFactory getHeaderFactory() {
        return headerFactory;
    }

    public Registrar getRegistrar() {
        return registrar;
    }


    public boolean isPresenceServer() {
        return configuration.enablePresenceServer;
    }

    public PresenceServer getPresenceServer() {
            return presenceServer;
    }

    public ProxyUtilities getProxyUtilities() {
        return proxyUtilities;
    }

    public SipStack getSipStack() {
        return sipStack;
    }

    public Configuration getConfiguration() {
        return configuration;
    }
    /** get the first allocated provider.
    */
    public SipProvider getSipProvider() {


        protected LinkedList listeningPoints;
        // Map the server transactions with the client transactions
        protected SipStack sipStack;
        protected SipProvider defaultProvider;

        protected MessageFactory messageFactory;
        protected HeaderFactory headerFactory;
        protected AddressFactory addressFactory;

        protected Configuration configuration;
        protected PresenceServer presenceServer;

        protected Registrar registrar;
        protected ProxyUtilities proxyUtilities;
        protected Authentication authentication;
        protected RequestForwarding requestForwarding;
        protected ResponseForwarding responseForwarding;


        public RequestForwarding getRequestForwarding() {
                return requestForwarding;
        }

        public ResponseForwarding getResponseForwarding() {
                return responseForwarding;
        }

        public AddressFactory getAddressFactory() {
                return addressFactory;
        }

        public MessageFactory getMessageFactory() {
                return messageFactory;
        }

        public HeaderFactory getHeaderFactory() {
                return headerFactory;
        }

        public Registrar getRegistrar() {
                return registrar;
        }


        public boolean isPresenceServer() {
                return configuration.enablePresenceServer;
        }

        public PresenceServer getPresenceServer() {
                return presenceServer;
        }

        public ProxyUtilities getProxyUtilities() {
                return proxyUtilities;
        }

        public SipStack getSipStack() {
                return sipStack;
        }

        public Configuration getConfiguration() {
                return configuration;
        }
        /** get the first allocated provider.
         */
        public SipProvider getSipProvider() {
                return this.defaultProvider;
    }

    public Authentication getAuthentication() {
        return authentication;
    }

    public boolean managesDomain( String domainAddress ) {
       return   configuration.hasDomain(domainAddress) ||
                registrar.hasRegistration("sip:"+domainAddress);
    }


    /** Creates new Proxy */
    public Proxy(String confFile) throws Exception{

        this.listeningPoints = new LinkedList();
        if (confFile==null) {
            System.out.println
            ("ERROR: Set the configuration file flag: " +
            "USE: -cf configuration_file_location.xml"  );
        }
        else {
            try {

                // First, let's parse the configuration file.
                ProxyConfigurationHandler handler=
                new ProxyConfigurationHandler(confFile);
                configuration=handler.getConfiguration();
                if (configuration==null ||
                !configuration.isValidConfiguration()) {
                    System.out.println
                    ("ERROR: the configuration file is not correct!"+
                    " Correct the errors first.");
                    throw new Exception
                    ("ERROR: the configuration file is not correct!"+
                    " Correct the errors first.");
                }
                else {

                    proxyUtilities=new ProxyUtilities(this);
                    presenceServer=new PresenceServer(this);
                    registrar=new Registrar(this);
                    requestForwarding=new RequestForwarding(this);
                    responseForwarding=new ResponseForwarding(this);
                }
            }
            catch (Exception ex) {
                System.out.println
                ("ERROR: exception raised while initializing the proxy");
                ex.printStackTrace();
                throw new Exception
                ("ERROR: exception raised while initializing the proxy");
            }
        }
    }



    /** This is a listener method.
     */
    public void processRequest(RequestEvent requestEvent) {
        Request request = requestEvent.getRequest();

        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        ServerTransaction serverTransaction=requestEvent.getServerTransaction();
        try {

            if (ProxyDebug.debug)
                ProxyDebug.println
                ("\n*****************************************************"+
                "\nRequest " + request.getMethod() +
                        " received:\n"+request.toString());


            if (ProxyDebug.debug)
                 ProxyUtilities.printTransaction(serverTransaction);



/*******************************************************************************/
/***********************  PROXY BEHAVIOR    ************************************/
/*******************************************************************************/

            /* RFC 3261: 16.2:
             * For all new requests, including any with unknown methods, an element
             * intending to proxy the request MUST:
             *
             * 1. Validate the request (Section 16.3)
             *
             * 2. Preprocess routing information (Section 16.4)
             *
             * 3. Determine target(s) for the request (Section 16.5)
             *
             * 4. Forward the request to each target (Section 16.6)
             *
             * 5. Process all responses (Section 16.7)
             */

/*******************************************************************************/
/***************************** 1. Validate the request (Section 16.3) **********/
/*******************************************************************************/

           /*
        }

        public Authentication getAuthentication() {
                return authentication;
        }

        public boolean managesDomain( String domainAddress ) {
                return   configuration.hasDomain(domainAddress) ||
                                registrar.hasRegistration("sip:"+domainAddress);
        }


        /** Creates new Proxy */
        public Proxy(String confFile) throws Exception{

                this.listeningPoints = new LinkedList();
                if (confFile==null) {
                        System.out.println
                        ("ERROR: Set the configuration file flag: " +
                                        "USE: -cf configuration_file_location.xml"  );
                }
                else {
                        try {

                                // First, let's parse the configuration file.
                                ProxyConfigurationHandler handler=
                                                new ProxyConfigurationHandler(confFile);
                                configuration=handler.getConfiguration();
                                if (configuration==null ||
                                                !configuration.isValidConfiguration()) {
                                        System.out.println
                                        ("ERROR: the configuration file is not correct!"+
                                                        " Correct the errors first.");
                                        throw new Exception
                                        ("ERROR: the configuration file is not correct!"+
                                                        " Correct the errors first.");
                                }
                                else {

                                        proxyUtilities=new ProxyUtilities(this);
                                        presenceServer=new PresenceServer(this);
                                        registrar=new Registrar(this);
                                        requestForwarding=new RequestForwarding(this);
                                        responseForwarding=new ResponseForwarding(this);
                                }
                        }
                        catch (Exception ex) {
                                System.out.println
                                ("ERROR: exception raised while initializing the proxy");
                                ex.printStackTrace();
                                throw new Exception
                                ("ERROR: exception raised while initializing the proxy");
                        }
                }
        }



        /** This is a listener method.
         */
        public void processRequest(RequestEvent requestEvent) {

                ProxyDebug.println("Pare to stack: "+Thread.currentThread().getStackTrace().toString());

                for (StackTraceElement ste : Thread.currentThread().getStackTrace()) {
                        ProxyDebug.println(ste.toString());
                }

                Request request = requestEvent.getRequest();

        ProxyDebug.println("perase sto 0 To request in proxy, request: "+request.toString());


                SipProvider sipProvider = (SipProvider) requestEvent.getSource();
                ServerTransaction serverTransaction=requestEvent.getServerTransaction();
                try {

                        if (ProxyDebug.debug)
                                ProxyDebug.println
                                ("\n****************************************************"+
                                                "\nRequest " + request.getMethod() +
                                                " received:\n"+request.toString());


                        if (ProxyDebug.debug)
                                ProxyUtilities.printTransaction(serverTransaction);



                        /*******************************************************************************/
                        /***********************  PROXY BEHAVIOR    ***********************************/
                        /*******************************************************************************/

                        /* RFC 3261: 16.2:
                         * For all new requests, including any with unknown methods, an element
                         * intending to proxy the request MUST:
                         *
                         * 1. Validate the request (Section 16.3)
                         *
                         * 2. Preprocess routing information (Section 16.4)
                         *
                         * 3. Determine target(s) for the request (Section 16.5)
                         *
                         * 4. Forward the request to each target (Section 16.6)
                         *
                         * 5. Process all responses (Section 16.7)
                         */

                        /*******************************************************************************/
                        /***************************** 1. Validate the request (Section 16.3) *********/
                        /*******************************************************************************/

                        /*
            Before an element can proxy a request, it MUST verify the message's
            validity
            */

            RequestValidation requestValidation=new RequestValidation(this);
            if ( !requestValidation.validateRequest
                (sipProvider,request,serverTransaction) ) {
                // An appropriate response has been sent back by the request
                // validation step, so we just return. The request has been
                // processed!
                if (ProxyDebug.debug)
                ProxyDebug.println
                ("Proxy, processRequest(), the request has not been"+
                " validated, so the request is discarded "  +
                " (an error code has normally been"+
                " sent back)");
                return;
            }

            // Let's check if the ACK is for the proxy: if there is no Route
            // header: it is mandatory for the ACK to be forwarded
            if ( request.getMethod().equals(Request.ACK) ) {
                  ListIterator routes = request.getHeaders(RouteHeader.NAME);
                  if (routes==null  || !routes.hasNext()) {
                      if (ProxyDebug.debug)
                    ProxyDebug.println("Proxy, processRequest(), "+
                    "the request is an ACK"+
                    " targeted for the proxy, we ignore it");
                    return;
                }
            }



            if (serverTransaction==null) {
                String method=request.getMethod();
                // Methods that creates dialogs, so that can
                // generate transactions
                if ( method.equals(Request.INVITE) ||
                     method.equals(Request.SUBSCRIBE)
                ) {
                    try{
                        serverTransaction=
                        sipProvider.getNewServerTransaction(request);
                         TransactionsMapping transactionsMapping=
                            (TransactionsMapping)
                            serverTransaction.getDialog().getApplicationData();
                         if (transactionsMapping == null) {
                             transactionsMapping =
                             new TransactionsMapping(serverTransaction);
                         }
                    }
                    catch(TransactionAlreadyExistsException e) {
                         if (ProxyDebug.debug)
                            ProxyDebug.println
                        ("Proxy, processRequest(), this request"+
                            " is a retransmission, we drop it!");
                    }
                }
           }

/*****************************************************************************/
/****** 2. Preprocess routing information (Section 16.4) *******************/
/*****************************************************************************/

            /*   The proxy MUST inspect the Request-URI of the request.  If the
```

```
                         */

                ProxyDebug.println("perase sto 1 To request in proxy, request: "+request.toString());


                        RequestValidation requestValidation=new RequestValidation(this);
                        if ( !requestValidation.validateRequest
                                        (sipProvider,request,serverTransaction) ) {
                                // An appropriate response has been sent back by the request
                                // validation step, so we just return. The request has been
                                // processed!
                                if (ProxyDebug.debug)
                                        ProxyDebug.println
                                        ("Proxy, processRequest(), the request has not been"+
                                                        " validated, so the request is discarded "  +
                                                        " (an error code has normally been"+
                                                        " sent back)");
                                return;
                        }

                        // Let's check if the ACK is for the proxy: if there is no Route
                        // header: it is mandatory for the ACK to be forwarded
                        if ( request.getMethod().equals(Request.ACK) ) {
                                ListIterator routes = request.getHeaders(RouteHeader.NAME);
                                if (routes==null  || !routes.hasNext()) {
                                        if (ProxyDebug.debug)
                                                ProxyDebug.println("Proxy, processRequest(), "+
                                                                "the request is an ACK"+
                                                                " targeted for the proxy, we ignore it");
                                        return;
                                }
                        }



                        if (serverTransaction==null) {
                                String method=request.getMethod();
                                // Methods that creates dialogs, so that can
                                // generate transactions
                                if ( method.equals(Request.INVITE) ||
                                                method.equals(Request.SUBSCRIBE)
                                                ) {
                                        try{
                                                serverTransaction=
                                                                sipProvider.getNewServerTransaction(request);
                                                TransactionsMapping transactionsMapping=
                                                                (TransactionsMapping)
                                                                serverTransaction.getDialog().getApplicationData();
                                                if (transactionsMapping == null) {
                                                        transactionsMapping =
                                                                        new TransactionsMapping(serverTransaction);
                                                }
                                        }
                                        catch(TransactionAlreadyExistsException e) {
                                                if (ProxyDebug.debug)
                                                        ProxyDebug.println
                                                        ("Proxy, processRequest(), this request"+
                                                                        " is a retransmission, we drop it!");
                                        }
                                }
                        }

                        /***************************************************************************/
                        /****** 2. Preprocess routing information (Section 16.4) *******************/
                        /***************************************************************************/
                ProxyDebug.println("perase sto 2 To request in proxy, request: "+request.toString());

                        /*   The proxy MUST inspect the Request-URI of the request.  If the
            Request-URI of the request contains a value this proxy previously
            placed into a Record-Route header field (see Section 16.6 item 4),
            the proxy MUST replace the Request-URI in the request with the last
@ Proxy.java:302 @ public class Proxy implements SipListener  {
            field value.  If it indicates this proxy, the proxy removes it
            from the Route header field (this route node has been
            reached).
           */

            ListIterator routes = request.getHeaders(RouteHeader.NAME);
            if (routes!=null) {
                if ( routes.hasNext() ) {
                    RouteHeader  routeHeader = (RouteHeader) routes.next();
                    Address routeAddress=routeHeader.getAddress();
                    SipURI routeSipURI=(SipURI)routeAddress.getURI();

                    String host = routeSipURI.getHost();
                    int port = routeSipURI.getPort();

                    if (sipStack.getIPAddress().equals(host) ) {
                        Iterator lps=sipStack.getListeningPoints();
                        while(lps!=null && lps.hasNext()) {
                            ListeningPoint lp=(ListeningPoint)lps.next();
                            if (lp.getPort()==port) {
                                if (ProxyDebug.debug)
                                    ProxyDebug.println
                                    ("Proxy, processRequest(),"+
                                    " we remove the first route form " +
                                    " the RouteHeader;"+
                                    " it matches the proxy");
                                routes.remove();
                                break;
                            }
                        }
                    }
                }
            }

            /*
                         */

                        ListIterator routes = request.getHeaders(RouteHeader.NAME);
                        if (routes!=null) {
                                if ( routes.hasNext() ) {
                                        RouteHeader  routeHeader = (RouteHeader) routes.next();
                                        Address routeAddress=routeHeader.getAddress();
                                        SipURI routeSipURI=(SipURI)routeAddress.getURI();

                                        String host = routeSipURI.getHost();
                                        int port = routeSipURI.getPort();

                                        if (sipStack.getIPAddress().equals(host) ) {
                                                Iterator lps=sipStack.getListeningPoints();
                                                while(lps!=null && lps.hasNext()) {
                                                        ListeningPoint lp=(ListeningPoint)lps.next();
                                                        if (lp.getPort()==port) {
                                                                if (ProxyDebug.debug)
                                                                        ProxyDebug.println
                                                                        ("Proxy, processRequest(),"+
                                                                                        " we remove the first route form " +
                                                                                        " the RouteHeader;"+
                                                                                        " it matches the proxy");
                                                                routes.remove();
                                                                break;
                                                        }
                                                }
                                        }
                                }
                        }

                        /*
            If the Request-URI contains a maddr parameter, the proxy MUST check
            to see if its value is in the set of addresses or domains the proxy
            is configured to be responsible for.  If the Request-URI has a maddr
@ Proxy.java:342 @ public class Proxy implements SipListener  {
            default) in the Request-URI, the proxy MUST strip the maddr and any
            non-default port or transport parameter and continue processing as if
            those values had not been present in the request.
             */
            URI requestURI=request.getRequestURI();
            if (requestURI.isSipURI()) {
                SipURI requestSipURI=(SipURI)requestURI;
                if (requestSipURI.getMAddrParam()!=null ) {
                    // The domain the proxy is configured to be responsible for is defined
                    // by stack_domain parameter in the configuration file:
                    if (configuration.hasDomain(requestSipURI.getMAddrParam())) {
                        if (ProxyDebug.debug)
                            ProxyDebug.println("Proxy, processRequest(),"+
                            " The maddr contains a domain we are responsible for,"+
                            " we remove the mAddr parameter from the original"+
                            " request");
                        // We have to strip the madr parameter:
                        requestSipURI.removeParameter("maddr");
                        // We have to strip the port parameter:
                        if (requestSipURI.getPort()!=5060 && requestSipURI.getPort()!=-1) {
                            requestSipURI.setPort(5060);
                        }
                        // We have to strip the transport parameter:
                        requestSipURI.removeParameter("transport");
                    }
                    else {
                        // The Maddr parameter is not a domain we have to take
                        // care of, we pass this check...
                    }
                }
                else {
                    // No Maddr parameter, we pass this check...
                }
            }
            else {
                // No SipURI, so no Maddr parameter, we pass this check...
            }



/*****************************************************************************/
/************* 3. Determine target(s) for the request (Section 16.5) **********/
/****************************************************************************/
            /*
             */

                        URI requestURI=request.getRequestURI();
                        if (requestURI.isSipURI()) {
                                SipURI requestSipURI=(SipURI)requestURI;
                                if (requestSipURI.getMAddrParam()!=null ) {
                                        // The domain the proxy is configured to be responsible for is defined
                                        // by stack_domain parameter in the configuration file:
                                        if (configuration.hasDomain(requestSipURI.getMAddrParam())) {
                                                if (ProxyDebug.debug)
                                                        ProxyDebug.println("Proxy, processRequest(),"+
                                                                        " The maddr contains a domain we are responsible for,"+
                                                                        " we remove the mAddr parameter from the original"+
                                                                        " request");
                                                // We have to strip the madr parameter:
                                                requestSipURI.removeParameter("maddr");
                                                // We have to strip the port parameter:
                                                if (requestSipURI.getPort()!=5060 && requestSipURI.getPort()!=-1) {
                                                        requestSipURI.setPort(5060);
                                                }
                                                // We have to strip the transport parameter:
                                                requestSipURI.removeParameter("transport");
                                        }
                                        else {
                                                // The Maddr parameter is not a domain we have to take
                                                // care of, we pass this check...
                                        }
                                }
                                else {
                                        // No Maddr parameter, we pass this check...
                                }
                        }
                        else {
                                // No SipURI, so no Maddr parameter, we pass this check...
                        }



                        /*****************************************************************************/
                        /************* 3. Determine target(s) for the request (Section 16.5) **********/
                        /****************************************************************************/
                        /*
            The set of targets will either be predetermined by the contents of the
            request or will be obtained from an abstract location service.  Each
            target in the set is represented as a URI.
             */

            Vector targetURIList=new Vector();
            URI targetURI;

           /* If the Request-URI of the request contains an maddr parameter, the
            * Request-URI MUST be placed into the target set as the only target
            * URI, and the proxy MUST proceed to Section 16.6.
            */

            if (requestURI.isSipURI()) {
                SipURI requestSipURI=(SipURI)requestURI;
                if (requestSipURI.getMAddrParam()!=null ) {
                    targetURI=requestURI;
                    targetURIList.addElement(targetURI);
                    if (ProxyDebug.debug)
                        ProxyDebug.println("Proxy, processRequest(),"+
                        " the only target is the Request-URI (mAddr parameter)");

                    // 4. Forward the request statefully:
                    requestForwarding.forwardRequest(targetURIList,sipProvider,
                                         request,serverTransaction,true);

                    return;
                }
            }

            /*
             */

                        Vector targetURIList=new Vector();
                        URI targetURI;

                        /* If the Request-URI of the request contains an maddr parameter, the
                         * Request-URI MUST be placed into the target set as the only target
                         * URI, and the proxy MUST proceed to Section 16.6.
                         */

                        if (requestURI.isSipURI()) {
                                SipURI requestSipURI=(SipURI)requestURI;
                                if (requestSipURI.getMAddrParam()!=null ) {
                                        targetURI=requestURI;
                                        targetURIList.addElement(targetURI);
                                        if (ProxyDebug.debug)
                                                ProxyDebug.println("Proxy, processRequest(),"+
                                                                " the only target is the Request-URI (mAddr parameter)");

                                        // 4. Forward the request statefully:
                                        requestForwarding.forwardRequest(targetURIList,sipProvider,
                                                        request,serverTransaction,true);

                                        return;
                                }
                        }

                        /*
              If the domain of the Request-URI indicates a domain this element is
              not responsible for, the Request-URI MUST be placed into the target
              set as the only target, and the element MUST proceed to the task of
              Request Forwarding (Section 16.6).
             */

            if (requestURI.isSipURI()) {
```

```
    SipURI requestSipURI=(SipURI)requestURI;
    if ( !configuration.hasDomain(requestSipURI.getHost() ) ) {
        if (ProxyDebug.debug)
            ProxyDebug.println("Proxy, processRequest(),"+
            " we are not responsible for the domain: Let's check if we have"+
            " a registration for this domain from another proxy");

        // We have to check if another proxy did not registered
        // to us, in this case we have to use the contacts provided
        // by the registered proxy to create the targets:
        if (registrar.hasDomainRegistered(request)) {
            targetURIList=registrar.getDomainContactsURI(request);
            if (targetURIList!=null && !targetURIList.isEmpty()) {
                if (ProxyDebug.debug) {
                    ProxyDebug.println("Proxy, processRequest(), we have"+
                    " a registration for this domain from another proxy");
                }
                // 4. Forward the request statefully:
                requestForwarding.forwardRequest(targetURIList,sipProvider,
                request,serverTransaction,true);
                return;

            }
            else {
                targetURIList=new Vector();
                ProxyDebug.println("Proxy, processRequest(),"+
                " we are not responsible for the domain: the only target"+
                " URI is given by the request-URI");
                targetURI=requestURI;
                targetURIList.addElement(targetURI);
            }
        }
        else {
            ProxyDebug.println("Proxy, processRequest(),"+
                    " we are not responsible for the domain: the only target"+
                    " URI is given by the request-URI");
            targetURI=requestURI;
            targetURIList.addElement(targetURI);
        }

        // 4. Forward the request statelessly:
        requestForwarding.forwardRequest(targetURIList,sipProvider,
        request,serverTransaction,false);

        return;
    }
    else {
            ProxyDebug.println("Proxy, processRequest(),"+
                    " we are responsible for the domain... Let's find the contact...");
    }
}

  // we use a SIP registrar:
 if ( request.getMethod().equals(Request.REGISTER) ) {
    if (ProxyDebug.debug)
            ProxyDebug.println("Incoming request Register");
    // we call the RegisterProcessing:
    registrar.processRegister
            (request,sipProvider,serverTransaction);
    //Henrik: let the presenceserver do some processing too
    if ( isPresenceServer()) {
        presenceServer.processRegisterRequest
            (sipProvider, request, serverTransaction);
    }
             */

    return;
 }

            if (requestURI.isSipURI()) {
                    SipURI requestSipURI=(SipURI)requestURI;
                    if ( !configuration.hasDomain(requestSipURI.getHost() ) ) {
                            if (ProxyDebug.debug)
                                    ProxyDebug.println("Proxy, processRequest(),"+
                                                    " we are not responsible for the domain: Let's check if we have"+
                                                    " a registration for this domain from another proxy");

                            // We have to check if another proxy did not registered
                            // to us, in this case we have to use the contacts provided
                            // by the registered proxy to create the targets:
                            if (registrar.hasDomainRegistered(request)) {
                                    targetURIList=registrar.getDomainContactsURI(request);
                                    if (targetURIList!=null && !targetURIList.isEmpty()) {
                                            if (ProxyDebug.debug) {
                                                    ProxyDebug.println("Proxy, processRequest(), we have"+
                                                                    " a registration for this domain from another proxy");
                                            }
                                            // 4. Forward the request statefully:
                                            requestForwarding.forwardRequest(targetURIList,sipProvider,
                                                            request,serverTransaction,true);
                                            return;

 /* If we receive a subscription targeted to a user that
  * is publishing its state here, send to presence server
  */
 if ( isPresenceServer() &&
    (request.getMethod().equals(Request.SUBSCRIBE))) {
     ProxyDebug.println("Incoming request Subscribe");

     if (presenceServer.isStateAgent(request)) {
         Request clonedRequest=(Request)request.clone();
         presenceServer.processSubscribeRequest(sipProvider,
                                                clonedRequest,
                                                serverTransaction);
     } else {
         // Do we know this guy?


         targetURIList = registrar.getContactsURI(request);
         if (targetURIList == null ) {
            // If not respond that we dont know him.
             ProxyDebug.println
            ("Proxy: received a Subscribe request to " +
             " a user in our domain that was not found, " +
             " responded 404");
             Response response=
                    messageFactory.createResponse
                    (Response.NOT_FOUND,request);
             if (serverTransaction!=null)
                 serverTransaction.sendResponse(response);
             else
                 sipProvider.sendResponse(response);
             return;
         } else  {
             ProxyDebug.println
                    ("Trying to forward subscribe to "
                    + targetURIList.toString() +
                    "\n" + request.toString());
             requestForwarding.forwardRequest
                    (targetURIList,sipProvider,
                    request,serverTransaction,false);

         }
     }
     return;
 }

 /** Received a Notify.
  *  TOADD: Check if it match active VirtualSubscriptions and update it
  **/
 if ( isPresenceServer() && (request.getMethod().equals
            (Request.NOTIFY) )) {
     System.out.println("Incoming request Notify");

     Response response=messageFactory.createResponse(481,request);
     response.setReasonPhrase("Subscription does not exist");
     if (serverTransaction!=null)
         serverTransaction.sendResponse(response);
         else
             sipProvider.sendResponse(response);
         ProxyDebug.println ("Proxy: received a Notify request. Probably wrong, responded 481");
         return;
     }


    if ( isPresenceServer() && ( request.getMethod().equalsIgnoreCase("PUBLISH"))) {

         System.out.println("Incoming request Publish");

         ProxyDebug.println("Proxy: received a Publish request.");
         Request clonedRequest=(Request)request.clone();


         if (presenceServer.isStateAgent(clonedRequest)) {
             ProxyDebug.println("PresenceServer.isStateAgent");
         } else {
             ProxyDebug.println("PresenceServer is NOT StateAgent");
         }

        if (presenceServer.isStateAgent(clonedRequest)) {
            presenceServer.processPublishRequest(sipProvider,
                                                 clonedRequest,
                                                 serverTransaction);
        } else {
            Response response=messageFactory.createResponse(Response.NOT_FOUND,request);
            if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else
                sipProvider.sendResponse(response);
        }
        return;
    }
                                        }
                                        else {
                                                targetURIList=new Vector();
                                                ProxyDebug.println("Proxy, processRequest(),"+
                                                                " we are not responsible for the domain: the only target"+
                                                                " URI is given by the request-URI");
                                                targetURI=requestURI;
                                                targetURIList.addElement(targetURI);
                                        }
                                }
                                else {
                                        ProxyDebug.println("Proxy, processRequest(),"+
                                                        " we are not responsible for the domain: the only target"+
                                                        " URI is given by the request-URI");
                                        targetURI=requestURI;
                                        targetURIList.addElement(targetURI);
                                }


                                // 4. Forward the request statelessly:
                                requestForwarding.forwardRequest(targetURIList,sipProvider,
                                                request,serverTransaction,false);

                                return;
                        }
                        else {
                                ProxyDebug.println("Proxy, processRequest(),"+
                                                " we are responsible for the domain... Let's find the contact...");
                        }
                }

                /*
                 * We insert the listener for the FORWARD and UNFORWARD requests here 22-1-2017
                 */


     // Forward to next hop but dont reply OK right away for the
  // BYE. Bye is end-to-end not hop by hop!
  if (request.getMethod().equals(Request.BYE) ) {
     if (serverTransaction == null) {
        if (ProxyDebug.debug)
            ProxyDebug.println
                ("Proxy, null server transactin for BYE");
          return;
        }
        Dialog d = serverTransaction.getDialog();
        TransactionsMapping transactionsMapping =
                (TransactionsMapping) d.getApplicationData();
        Dialog peerDialog = (Dialog) transactionsMapping.getPeerDialog
                (serverTransaction);
        Request clonedRequest = (Request) request.clone();
        FromHeader from = (FromHeader)
                clonedRequest.getHeader(FromHeader.NAME);
        from.removeParameter("tag");
        ToHeader to = (ToHeader)
                clonedRequest.getHeader(ToHeader.NAME);
        to.removeParameter("tag");
        ViaHeader via = this.getStackViaHeader();
        clonedRequest.addHeader(via);
       if ( peerDialog.getState() != null ) {
          ClientTransaction newct =
                        sipProvider.getNewClientTransaction
                        (clonedRequest);
          transactionsMapping.addMapping(serverTransaction,newct);
          peerDialog.sendRequest(newct);
          return;
        } else {
          // the peer dialog is not yet established so bail out.
          // this is a client error - client is sending BYE
          // before dialog establishment.
          if (ProxyDebug.debug)
              ProxyDebug.println
                ("Proxy, bad dialog state - BYE dropped");
          return;
       }
}
        ProxyDebug.println("perase to 3  request: "+request.toString());
        //ProxyDebug.println("method for request FORWARD and UNFORWARD "+request.getMethod().toString());
                if (request.getMethod().equals("FORWARD") || request.getMethod().equals("UNFORWARD")){
                ProxyDebug.println("UPDATE request in proxy, request: "+request.toString());


                //send the response to the user

                registrar.processUserForward(request,sipProvider,serverTransaction);


                return;
                }




    /*

                /*
                 * We insert the listener for the BLOCK and UNBLOCK requests here 25-1-2017
                 */
        //ProxyDebug.println("method for request BLOCK or UNBLOCK "+request.getMethod().toString());
                if (request.getMethod().equals("BLOCK") || request.getMethod().equals("UNBLOCK") )
                {
                ProxyDebug.println("BLOCK request in proxy, request: "+request.toString());

                //send the response to the user

                registrar.processUserBlocking(request,sipProvider,serverTransaction);

                return;
                }

                /*
                 * We insert the listener for the INFO request in order to give the user his
                 * Blocked Users List and his Forward callee if need be by the GUI
```

```
 */
if (request.getMethod().equals("INFO") )
{
ProxyDebug.println("INFO request in proxy, request: "+request.toString());

//send the response to the user

registrar.processUserInfo(request,sipProvider,serverTransaction, headerFactory);

return;
}


/**
 * If the request is options and the content says duration 29-1-2017
 */


if (request.getMethod().equals("OPTIONS")){
        String content = new String( request.getRawContent());
        if (content.contains("Duration")){

                ProxyDebug.println("OPTION request in proxy , request: "+request.toString());

        //send the response to the user

        registrar.processUserBilling(request,sipProvider,serverTransaction, headerFactory);


        return;
        }


}

// we use a SIP registrar:
if ( request.getMethod().equals(Request.REGISTER) ) {
        if (ProxyDebug.debug)
                ProxyDebug.println("Incoming request Register");
        // we call the RegisterProcessing:
        registrar.processRegister
        (request,sipProvider,serverTransaction);
        //Henrik: let the presenceserver do some processing too
        if ( isPresenceServer()) {
                presenceServer.processRegisterRequest
                (sipProvider, request, serverTransaction);
        }

        return;
}



/* If we receive a subscription targeted to a user that
 * is publishing its state here, send to presence server
 */
if ( isPresenceServer() &&
                (request.getMethod().equals(Request.SUBSCRIBE))) {
        ProxyDebug.println("Incoming request Subscribe");

        if (presenceServer.isStateAgent(request)) {
                Request clonedRequest=(Request)request.clone();
                presenceServer.processSubscribeRequest(sipProvider,
                                clonedRequest,
                                serverTransaction);
        } else {
                // Do we know this guy?


                targetURIList = registrar.getContactsURI(request);
                if (targetURIList == null ) {
                        // If not respond that we dont know him.
                        ProxyDebug.println
                        ("Proxy: received a Subscribe request to " +
                                        " a user in our domain that was not found, " +
                                        " responded 404");
                        Response response=
                                        messageFactory.createResponse
                                        (Response.NOT_FOUND,request);
                        if (serverTransaction!=null)
                                serverTransaction.sendResponse(response);
                        else
                                sipProvider.sendResponse(response);
                        return;
                } else  {
                        ProxyDebug.println
                        ("Trying to forward subscribe to "
                                        + targetURIList.toString() +
                                        "\n" + request.toString());
                        requestForwarding.forwardRequest
                        (targetURIList,sipProvider,
                                        request,serverTransaction,false);

                }
        }
        return;
}

/** Received a Notify.
 *  TOADD: Check if it match active VirtualSubscriptions and update it
 **/
if ( isPresenceServer() && (request.getMethod().equals
                (Request.NOTIFY) )) {
        System.out.println("Incoming request Notify");

        Response response=messageFactory.createResponse(481,request);
        response.setReasonPhrase("Subscription does not exist");
        if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
        else
                sipProvider.sendResponse(response);
        ProxyDebug.println ("Proxy: received a Notify request. Probably wrong, responded 481");
        return;
}



if ( isPresenceServer() && ( request.getMethod().equalsIgnoreCase("PUBLISH"))) {

        System.out.println("Incoming request Publish");

        ProxyDebug.println("Proxy: received a Publish request.");
        Request clonedRequest=(Request)request.clone();


        if (presenceServer.isStateAgent(clonedRequest)) {
                ProxyDebug.println("PresenceServer.isStateAgent");
        } else {
                ProxyDebug.println("PresenceServer is NOT StateAgent");
        }

        if (presenceServer.isStateAgent(clonedRequest)) {
                presenceServer.processPublishRequest(sipProvider,
                                clonedRequest,
                                serverTransaction);
        } else {
                Response response=messageFactory.createResponse(Response.NOT_FOUND,request);
                if (serverTransaction!=null)
                        serverTransaction.sendResponse(response);
                else
                        sipProvider.sendResponse(response);
        }
        return;
}
```

```
                        // Forward to next hop but dont reply OK right away for the
                        // BYE. Bye is end-to-end not hop by hop!
                        if (request.getMethod().equals(Request.BYE) ) {
                                if (serverTransaction == null) {
                                        if (ProxyDebug.debug)
                                                ProxyDebug.println
                                                ("Proxy, null server transactin for BYE");
                                        return;
                                }
                                Dialog d = serverTransaction.getDialog();
                                TransactionsMapping transactionsMapping =
                                                (TransactionsMapping) d.getApplicationData();
                                Dialog peerDialog = (Dialog) transactionsMapping.getPeerDialog
                                                (serverTransaction);
                                Request clonedRequest = (Request) request.clone();
                                FromHeader from = (FromHeader)
                                                clonedRequest.getHeader(FromHeader.NAME);
                                from.removeParameter("tag");
                                ToHeader to = (ToHeader)
                                                clonedRequest.getHeader(ToHeader.NAME);
                                to.removeParameter("tag");
                                ViaHeader via = this.getStackViaHeader();
                                clonedRequest.addHeader(via);
                                if ( peerDialog.getState() != null ) {
                                        ClientTransaction newct =
                                                        sipProvider.getNewClientTransaction
                                                        (clonedRequest);
                                        transactionsMapping.addMapping(serverTransaction,newct);
                                        peerDialog.sendRequest(newct);
                                        return;
                                } else {
                                        // the peer dialog is not yet established so bail out.
                                        // this is a client error - client is sending BYE
                                        // before dialog establishment.
                                        if (ProxyDebug.debug)
                                                ProxyDebug.println
                                                ("Proxy, bad dialog state - BYE dropped");
                                        return;
                                }
                        }



                        /*
              If the target set for the request has not been predetermined as
              described above, this implies that the element is responsible for the
              domain in the Request-URI, and the element MAY use whatever mechanism
@ Proxy.java:713 @ public class Proxy implements SipListener  {
              When accessing the location service constructed by a registrar, the
              Request-URI MUST first be canonicalized as described in Section 10.3
              before being used as an index.
             */
             if (requestURI.isSipURI()) {
                SipURI requestSipURI=(SipURI)requestURI;
                Iterator iterator=requestSipURI.getParameterNames();
                if (ProxyDebug.debug)
                    ProxyDebug.println("Proxy, processRequest(), we canonicalized"+
                    " the request-URI");
                while (iterator!=null && iterator.hasNext()) {
                    String name=(String)iterator.next();
                    requestSipURI.removeParameter(name);
                }
             }



            if ( registrar.hasRegistration(request)  ) {

                targetURIList=registrar.getContactsURI(request);

                // We fork only INVITE
                if (targetURIList!=null && targetURIList.size()>1
                                && !request.getMethod().equals("INVITE") ) {
                        if (ProxyDebug.debug)
                                ProxyDebug.println
                                ("Proxy, processRequest(), the request "+
                                                " to fork is not an INVITE, so we will process"+
                                " it with the first target as the only target.");
                        targetURI= (URI)targetURIList.firstElement();
                        targetURIList=new Vector();
                        targetURIList.addElement(targetURI);
                        // 4. Forward the request statefully to the target:
                        requestForwarding.forwardRequest(targetURIList,sipProvider,
                                        request,serverTransaction,true);
                        return;
                }

                if (targetURIList!=null && !targetURIList.isEmpty()) {
                        if (ProxyDebug.debug)
                                ProxyDebug.println
                                ("Proxy, processRequest(), the target set"+
                                                " is the set of the contacts URI from the " +
                                " location service");
                /*
                        //  ECE355 Changes - Aug. 2005.
                        // Call registry  service, get response (uri - wsdl).
                        // if response is not null then
                        //    do our staff
                        //    send to caller decline message by building a decline msg
                        //    and attach wsdl uri  in the message body
                        // else .. continue the logic below ...


                        // Lets assume that wsdl_string contains the message with all the
                        // service names and uri's for each service in the required format

                    // Query for web services for the receiver of INVITE
                        //  Use WebServices class to get services for org

                        String messageBody = "" ;
                        WebServicesQuery wsq = null ;
                        wsq  = WebServicesQuery.getInstance();

                        //  Get services info for receiver
                        //  A receiver is represented as an organization in the Service Registry

                    To to = (To)request.getHeader(ToHeader.NAME);
                        String toAddress = to.getUserAtHostPort();

                        // Remove all characters after the @ sign from To address
                        StringBuffer sb = new StringBuffer(toAddress);
                        int endsAt = sb.indexOf("@");
                        String orgNamePattern = sb.substring(0, endsAt);


                        Collection serviceInfoColl = wsq.findServicesForOrg(orgNamePattern);

                        // If services are found for this receiver (Org), build DECLINE message and
                        // send to client
                        if (serviceInfoColl != null) {
                        if (serviceInfoColl.size()!= 0 ){
                                System.out.println("Found " + serviceInfoColl.size() + " services for o rg " + orgNamePattern) ;
                                // Build message body for DECLINE message with Service Info
                                messageBody = serviceInfoColl.size()+ " -- " ;

                                        Iterator servIter = serviceInfoColl.iterator();
                                while (servIter.hasNext()) {
                                        ServiceInfo servInfo = (ServiceInfo)servIter.next();
                                        messageBody =  messageBody  + servInfo.getDescription()+ " " + servInfo.getWsdluri() + " " + servInfo.getEndPoint()+ " -- ";


                                        System.out.println("Name: " + servInfo.getName()) ;
                                        System.out.println("Providing Organization: " + servInfo.getProvidingOrganization()) ;
                                        System.out.println("Description: " + servInfo.getDescription()) ;
                                        System.out.println("Service End Point " + servInfo.getEndPoint()) ;
                                        System.out.println("wsdl wri " + servInfo.getWsdluri()) ;
```

```
                            System.out.println("--------------------------------");


                    }

                    System.out.println("ServiceInfo - Message Body  " + messageBody) ;

                    // Build and send DECLINE message with web service info

                    ContentTypeHeader contentTypeHeader = new ContentType(
                                    "text", "plain");

                    Response response = messageFactory.createResponse(
                                    Response.DECLINE, request, contentTypeHeader,
                                    messageBody);



                    if (serverTransaction != null)
                            serverTransaction.sendResponse(response);
                    else
                            sipProvider.sendResponse(response);
                    return;
            }
            else
                    System.out.println("There are no services for org " + orgNamePattern) ;

            }

            // End of ECE355 change

             */

            // 4. Forward the request statefully to each target Section 16.6.:
            requestForwarding.forwardRequest
            (targetURIList,sipProvider,
                            request,serverTransaction,true);

            return;
    } else {
            // Let's continue and try the default hop.
    }
}

 // The registrar cannot help to decide the targets, so let's use
 // our router: the default hop!
ProxyDebug.println
    ("Proxy, processRequest(), the registrar cannot help"+
" to decide the targets, so let's use our router: the default hop");
Router router=sipStack.getRouter();
if (router!=null) {
    ProxyHop hop =(ProxyHop) router.getOutboundProxy();
    if (hop!=null ) {
        if (ProxyDebug.debug)
            ProxyDebug.println
            ("Proxy, processRequest(), the target set"+
            " is the defaut hop: outbound proxy");

        // Bug fix contributed by Joe Provino
        String user = null;

        if (requestURI.isSipURI()) {
            SipURI requestSipURI=(SipURI)requestURI;
            user = requestSipURI.getUser();
        }

        SipURI hopURI=addressFactory.createSipURI
            (user,hop.getHost());
        hopURI.setTransportParam(hop.getTransport());
        hopURI.setPort(hop.getPort());
        targetURI=hopURI;
        targetURIList.addElement(targetURI);

        // 4. Forward the request statelessly to each target Section 16.6.:
        requestForwarding.forwardRequest(targetURIList,sipProvider,
        request,serverTransaction,false);

        return;
    }
}

/* If the target set remains empty after applying all of the above, the
             */
            if (requestURI.isSipURI()) {
                    SipURI requestSipURI=(SipURI)requestURI;
                    Iterator iterator=requestSipURI.getParameterNames();
                    if (ProxyDebug.debug)
                            ProxyDebug.println("Proxy, processRequest(), we canonicalized"+
                                            " the request-URI");
                    while (iterator!=null && iterator.hasNext()) {
                            String name=(String)iterator.next();
                            requestSipURI.removeParameter(name);
                    }
            }

            ProxyDebug.println("krakovia sipProvider: "+sipProvider.toString());
            ProxyDebug.println("krakovia request: "+request.toString());
            ProxyDebug.println("krakovia serverTransaction: "+request.getRequestURI());


            //Check the Blocked User List for the specific request
            //if the user is blocked then it should return busy
            String caller = request.getHeader(FromHeader.NAME).toString();
            String callee = request.getHeader(ToHeader.NAME).toString();
            caller = caller.substring(caller.indexOf("<") + 1, caller.indexOf(">"));
            callee = callee.substring(callee.indexOf("<") + 1, callee.indexOf(">"));
            caller = caller.split(";")[0];

            ProxyDebug.println("Caller and Callee: "+caller +"~"+callee);
            boolean userIsBlocked = registrar.foundInBlockedUsersList(callee, caller);
            if (userIsBlocked){
                    ProxyDebug.println
                    ("Proxy: User: "+callee+" is Already Blocked from: "+caller);
                    Response response=
                                    messageFactory.createResponse
                                    (Response.NOT_FOUND,request);
                    if (serverTransaction!=null)
                            serverTransaction.sendResponse(response);
                    else
                            sipProvider.sendResponse(response);
                    return;
            }

            //Check if the caller is the same as the final forwardee
            ProxyDebug.println("Check if the caller is the same as the final forwardee");
            String key=registrar.getKey(request);
            String finalForwardee = registrar.findFinalForwardee(key);

            if (finalForwardee != null && finalForwardee.equals(caller)) {
                    ProxyDebug.println("finalforwardee is the same as the caller");
                    Response response=messageFactory.createResponse
                                    (Response.BAD_REQUEST,request);
                    if (serverTransaction!=null)
                            serverTransaction.sendResponse(response);
                    else sipProvider.sendResponse(response);
                    if (ProxyDebug.debug) {
                            ProxyDebug.println
                            ("Registrar, FinalForwardee same as the "
                                            + "caller Cycle will be created, response sent:");
                            ProxyDebug.print(response.toString());
                    }


                    return;
            }

            /**
```

```
                 * Changes request based on final forwardee
                 */
                Request newrequest = transformRequestBasedOnForwardings(request);

                if (newrequest == null){
                        ProxyDebug.println
                        ("Proxy: User To forward not found online");
                        Response response=
                                        messageFactory.createResponse
                                        (Response.NOT_FOUND,request);
                        if (serverTransaction!=null)
                                serverTransaction.sendResponse(response);
                        else
                                sipProvider.sendResponse(response);
                        return;
                }
                request = newrequest;

                if ( registrar.hasRegistration(request)  ) {

                        targetURIList=registrar.getContactsURI(request);

                        // We fork only INVITE
                        if (targetURIList!=null && targetURIList.size()>1
                                        && !request.getMethod().equals("INVITE") ) {
                                if (ProxyDebug.debug)
                                        ProxyDebug.println
                                        ("Proxy, processRequest(), the request "+
                                                        " to fork is not an INVITE, so we will process"+
                                                        " it with the first target as the only target.");
                                targetURI= (URI)targetURIList.firstElement();
                                targetURIList=new Vector();
                                targetURIList.addElement(targetURI);
                                // 4. Forward the request statefully to the target:
                                requestForwarding.forwardRequest(targetURIList,sipProvider,
                                                request,serverTransaction,true);
                                return;
                        }

                        if (targetURIList!=null && !targetURIList.isEmpty()) {
                                if (ProxyDebug.debug)
                                        ProxyDebug.println
                                        ("Proxy, processRequest(), the target set"+
                                                        " is the set of the contacts URI from the " +
                                                        " location service");

                                // 4. Forward the request statefully to each target Section 16.6.:

                                /*
                                ProxyDebug.println("krakovia target URI LIST: "+targetURIList.toString());
                                ProxyDebug.println("krakovia sipProvider: "+sipProvider.toString());
                                ProxyDebug.println("krakovia request: "+request.toString());
                                ProxyDebug.println("krakovia serverTransaction: "+request.getRequestURI());
                                */

                                requestForwarding.forwardRequest
                                (targetURIList,sipProvider,
                                                request,serverTransaction,true);

                                /*
                                 * Here we have an invite request and we should run a method which runs in a thread
                                 * and polls over the users in order to charge the caller adeptly
                                 * Pathological Scenario
                                 */
                                /*
                                new Thread(() -> {
                                        this.callPolling( caller, callee);
                                }).start();
                                */

                                return;
                        } else {
                                // Let's continue and try the default hop.
                        }
                }

                // The registrar cannot help to decide the targets, so let's use
                // our router: the default hop!
                ProxyDebug.println
                ("Proxy, processRequest(), the registrar cannot help"+
                                " to decide the targets, so let's use our router: the default hop");
                Router router=sipStack.getRouter();
                if (router!=null) {
                        ProxyHop hop =(ProxyHop) router.getOutboundProxy();
                        if (hop!=null ) {
                                if (ProxyDebug.debug)
                                        ProxyDebug.println
                                        ("Proxy, processRequest(), the target set"+
                                                        " is the defaut hop: outbound proxy");

                                // Bug fix contributed by Joe Provino
                                String user = null;

                                if (requestURI.isSipURI()) {
                                        SipURI requestSipURI=(SipURI)requestURI;
                                        user = requestSipURI.getUser();
                                }

                                SipURI hopURI=addressFactory.createSipURI
                                                (user,hop.getHost());
                                hopURI.setTransportParam(hop.getTransport());
                                hopURI.setPort(hop.getPort());
                                targetURI=hopURI;
                                targetURIList.addElement(targetURI);

                                // 4. Forward the request statelessly to each target Section 16.6.:
                                requestForwarding.forwardRequest(targetURIList,sipProvider,
                                                request,serverTransaction,false);

                                return;
                        }
                }

                /* If the target set remains empty after applying all of the above, the
         proxy MUST return an error response, which SHOULD be the 480
         (Temporarily Unavailable) response.
     */
     Response response=messageFactory.createResponse
     (Response.TEMPORARILY_UNAVAILABLE,request);
     if (serverTransaction!=null)
            serverTransaction.sendResponse(response);
     else sipProvider.sendResponse(response);

     if (ProxyDebug.debug)
         ProxyDebug.println("Proxy, processRequest(), unable to set "+
         " the targets, 480 (Temporarily Unavailable) replied:\n"+
         response.toString() );

}
catch (Exception ex){
    try{
        if (ProxyDebug.debug) {
            ProxyDebug.println("Proxy, processRequest(), internal error, "+
            "exception raised:");
            ProxyDebug.logException(ex);
            ex.printStackTrace();
        }

        // This is an internal error:
        // Let's return a 500 SERVER_INTERNAL_ERROR
        Response response=messageFactory.createResponse
        (Response.SERVER_INTERNAL_ERROR,request);
        if (serverTransaction!=null)
            serverTransaction.sendResponse(response);
        else sipProvider.sendResponse(response);

        if (ProxyDebug.debug)
            ProxyDebug.println("Proxy, processRequest(),"+
                    " 500 SERVER_INTERNAL_ERROR replied:\n"+
                    response.toString());
            }
            catch (Exception e){
                e.printStackTrace();
            }
        }
    }

    /** This is a listener method.
     */
    public void processResponse(ResponseEvent responseEvent) {
        try{


            Response response = responseEvent.getResponse();
            SipProvider sipProvider = (SipProvider) responseEvent.getSource();
            ClientTransaction clientTransaction=responseEvent.getClientTransaction();

            ProxyDebug.println
            ("\n***************************************************************"+
            "\n***************************************************************"+
            "\nResponse "+response.getStatusCode() + " "+response.getReasonPhrase()
            +" received:\n"+response.toString() );
            ProxyDebug.println("Processing Response in progress");

            if (ProxyDebug.debug)
                ProxyUtilities.printTransaction(clientTransaction);

            //Henrik - added handling of responses addressed to server
            //If we use a presenceserver, and if statuscode was OK...
            CSeqHeader cseqHeader = (CSeqHeader)response.getHeader(CSeqHeader.NAME);

             if (cseqHeader.getMethod().equals("SUBSCRIBE")) {
                    presenceServer.processSubscribeResponse((Response)response.clone(), clientTransaction);
             } else if (cseqHeader.getMethod().equals("NOTIFY")) {
                    //presenceServer.processNotifyResponse((Response)response.clone(), clientTransaction);
             }

            responseForwarding.forwardResponse(sipProvider, response,clientTransaction);

        } catch (Exception ex) {
            if (ProxyDebug.debug) {
                ProxyDebug.println("Proxy, processResponse(), internal error, "+
                "exception raised:");
                ProxyDebug.logException(ex);
            }
        }
    }


    /** JAIN Listener method.
     */
    public void processTimeout(TimeoutEvent timeOutEvent) {
        ProxyDebug.println("TimeoutEvent received");
        SipProvider sipProvider = (SipProvider)timeOutEvent.getSource();
        TransactionsMapping transactionsMapping = null;
        if (timeOutEvent.isServerTransaction()) {
            ServerTransaction serverTransaction  =
            timeOutEvent.getServerTransaction();
            Dialog dialog = serverTransaction.getDialog();
            if (dialog != null) {
                transactionsMapping =
                        (TransactionsMapping) dialog.getApplicationData();
                transactionsMapping.removeMapping(serverTransaction);
            }
        } else {
            ClientTransaction clientTransaction =
            timeOutEvent.getClientTransaction();
            Dialog dialog = clientTransaction.getDialog();
            ServerTransaction st = null;
            if (dialog != null) {
                transactionsMapping =
                (TransactionsMapping) dialog.getApplicationData();
                if (transactionsMapping != null)  {
                   st = transactionsMapping.getServerTransaction
                   (clientTransaction);
                         */
                        Response response=messageFactory.createResponse
                                        (Response.TEMPORARILY_UNAVAILABLE,request);
                        if (serverTransaction!=null)
                                serverTransaction.sendResponse(response);
                        else sipProvider.sendResponse(response);

                        if (ProxyDebug.debug)
                                ProxyDebug.println("Proxy, processRequest(), unable to set "+
                                                " the targets, 480 (Temporarily Unavailable) replied:\n"+
                                                response.toString() );

                }
                catch (Exception ex){
                        try{
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println("Proxy, processRequest(), internal error, "+
                                                        "exception raised:");
                                        ProxyDebug.logException(ex);
                                        ex.printStackTrace();
                                }

                                // This is an internal error:
                                // Let's return a 500 SERVER_INTERNAL_ERROR
                                Response response=messageFactory.createResponse
                                                (Response.SERVER_INTERNAL_ERROR,request);
                                if (serverTransaction!=null)
                                        serverTransaction.sendResponse(response);
                                else sipProvider.sendResponse(response);

                                if (ProxyDebug.debug)
                                        ProxyDebug.println("Proxy, processRequest(),"+
                                                        " 500 SERVER_INTERNAL_ERROR replied:\n"+
                                                        response.toString());
                        }
                        catch (Exception e){
                                e.printStackTrace();
                        }
                }
               if (st==null) {
                  ProxyDebug.println
                  ("ERROR, Unable to retrieve the server transaction,"+
                  " cannot process timeout!");
                  return;
               }
            } else {
               ProxyDebug.println
                  ("ERROR, Unable to retrieve the transaction Mapping,"+
                  " cannot process timeout!");
                  return;
            }
            Request request = st.getRequest();
            // This removes the given mapping from the table but not
            // necessarily the whole thing.
            transactionsMapping.removeMapping(clientTransaction);
            if (!transactionsMapping.hasMapping(st)) {
                // No more mappings left in the transaction table.
                try {
                    Response response = messageFactory.createResponse
                    (Response.REQUEST_TIMEOUT, request);
                    st.sendResponse(response);
                } catch (ParseException ex) {
                    ex.printStackTrace();
                } catch (SipException ex1) {
                    ex1.printStackTrace();
                }
            }
        }


    }
```

```
/***********************  Methods for         ***************
 *    starting and stopping the proxy                       *
 ************************************************************/

/** Start the proxy, this method has to be called after the init method
 * throws Exception that which can be caught by the upper application
 */
public void start() throws Exception {
    if (configuration!=null
    && configuration.isValidConfiguration()) {
        Properties properties=new Properties();
        // LOGGING property:

        if (configuration.enableDebug) {
            ProxyDebug.debug=true;
            ProxyDebug.setProxyOutputFile(configuration.outputProxy);
            ProxyDebug.println("DEBUG properties set!");
            if (configuration.badMessageLogFile!=null)
                properties.setProperty("gov.nist.javax.sip.BAD_MESSAGE_LOG",
                configuration.badMessageLogFile);
            if (configuration.debugLogFile!=null) {
                properties.setProperty("gov.nist.javax.sip.DEBUG_LOG",
                configuration.debugLogFile);

            }
            if (configuration.serverLogFile!=null)
                properties.setProperty("gov.nist.javax.sip.SERVER_LOG",
                configuration.serverLogFile);
            if (configuration.debugLogFile != null)
                properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL",
                "32");
            else
            if (configuration.serverLogFile != null)
                properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL",
                "16");
            else
                properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL",
                "0");


        }
        else {
            System.out.println("DEBUG properties not set!");
        }
         registrar.setExpiresTime(configuration.expiresTime);

        // STOP TIME
        if (configuration.stopTime!=null) {
            try {
                long stopTime=Long.parseLong(configuration.stopTime);
                StopProxy stopProxy=new StopProxy(this);
                Timer timer=new Timer();
                timer.schedule(stopProxy,stopTime);
            }
            catch(Exception e) {
                e.printStackTrace();
            }
        }

        sipStack = null;

        SipFactory sipFactory = SipFactory.getInstance();
        sipFactory.setPathName("gov.nist");

        headerFactory = sipFactory.createHeaderFactory();
        addressFactory = sipFactory.createAddressFactory();
        messageFactory = sipFactory.createMessageFactory();


        // Create SipStack object

        properties.setProperty("javax.sip.IP_ADDRESS",
        configuration.stackIPAddress);

        // We have to add the IP address of the proxy for the domain:
        configuration.domainList.addElement(configuration.stackIPAddress);
        ProxyDebug.println("The proxy is responsible for the domain:"+configuration.stackIPAddress);

        properties.setProperty("javax.sip.STACK_NAME",
        configuration.stackName);
        if (configuration.check(configuration.outboundProxy))
            properties.setProperty("javax.sip.OUTBOUND_PROXY",
            configuration.outboundProxy);
        if (configuration.check(configuration.routerPath))
            properties.setProperty("javax.sip.ROUTER_PATH",
            configuration.routerPath);
        if (configuration.check(configuration.extensionMethods))
            properties.setProperty("javax.sip.EXTENSION_METHODS",
            configuration.extensionMethods);
        // This has to be hardcoded to true. for the proxy.
        properties.setProperty("javax.sip.RETRANSMISSION_FILTER",
            "on");

        if (configuration.check(configuration.maxConnections) )
            properties.setProperty("gov.nist.javax.sip.MAX_CONNECTIONS",
            configuration.maxConnections);
        if (configuration.check(configuration.maxServerTransactions) )
            properties.setProperty("gov.nist.javax.sip.MAX_SERVER_TRANSACTIONS",
            configuration.maxServerTransactions);
        if (configuration.check(configuration.threadPoolSize) )
            properties.setProperty("gov.nist.javax.sip.THREAD_POOL_SIZE",
            configuration.threadPoolSize);

        if (configuration.domainList!=null)
        for ( int j=0;j<configuration.domainList.size();j++) {
            String domain=(String)configuration.domainList.elementAt(j);
            ProxyDebug.println("Here is one domain to take care of:"+domain);
        }
        else ProxyDebug.println("No domain to take care of...");

        if (configuration.accessLogViaRMI) {
            properties.setProperty("gov.nist.javax.sip.ACCESS_LOG_VIA_RMI",
            "true");

            properties.setProperty("gov.nist.javax.sip.RMI_PORT",
            configuration.logRMIPort);

            if (configuration.check(configuration.logLifetime) )
                properties.setProperty("gov.nist.javax.sip.LOG_LIFETIME",
                configuration.logLifetime);

        }

        sipStack = sipFactory.createSipStack(properties);



        // Authentication part:
        if (configuration.enableAuthentication) {
            authentication =new Authentication(this);
            try{

                Class authMethodClass =
                            Class.forName(configuration.classFile);
                AuthenticationMethod authMethod
                = (AuthenticationMethod)
                authMethodClass.newInstance();
                authMethod.initialize(configuration.passwordsFile);

                authentication.setAuthenticationMethod(authMethod);

            }
            catch(Exception e) {
                    ProxyDebug.println
                    ("ERROR, authentication process stopped, exception raised:");
                    e.printStackTrace();
                }
            }

            // We create the Listening points:
            Vector lps=configuration.getListeningPoints();

            for ( int i=0;lps!=null && i<lps.size();i++) {
                Association a=(Association)lps.elementAt(i);
                try{
                    System.out.println("transport  " + a.transport);
                    System.out.println("port  " +
                    Integer.valueOf(a.port).intValue());
                    ListeningPoint lp=sipStack.createListeningPoint
                    (Integer.valueOf(a.port).intValue(),
                    a.transport);
                    this.listeningPoints.add(lp);
                    SipProvider sipProvider = sipStack.createSipProvider(lp);
                    if (this.defaultProvider == null)
                        this.defaultProvider = sipProvider;
                    sipProvider.addSipListener( this );
                }
                catch(Exception e) {
                    e.printStackTrace();
                    ProxyDebug.println
                    ("ERROR: listening point not created ");
                }
            }
            // Export the registry for polling by the responder.

            if (configuration.exportRegistry )
                // initialize the registrar for  RMI.
                this.registrar.initRMIBindings();


            // Parse static configuration for registrar.
            if (configuration.enableRegistrations) {
                String value=configuration.registrationsFile;
                ProxyDebug.println("Parsing the XML registrations file: "+value);
                if (value==null || value.trim().equals("")) {
                    ProxyDebug.println("You have to set the registrations file...");
                } else {
                    registrar.parseXMLregistrations(value);
                }
            }
            else ProxyDebug.println("No registrations to parse...");

            // Register to proxies if any:
            registrar.registerToProxies();

        }
        else {
            System.out.println("ERROR: the configuration file is not correct!"+
            " Correct the errors first.");
        }
    }

  /** Stop the proxy, this method has to be called after the start method
     * throws Exception that which can be caught by the upper application
     */
    public void stop()  throws Exception {
        if (sipStack==null) return;
        this.presenceServer.stop();

        Iterator sipProviders=sipStack.getSipProviders();
        if (sipProviders!=null) {
            while( sipProviders.hasNext()) {
                SipProvider sp=(SipProvider)sipProviders.next();
                sp.removeSipListener(this);
                sipStack.deleteSipProvider(sp);
                sipProviders=sipStack.getSipProviders();
                System.out.println("One sip Provider removed!");
            }
        }
        else {
            ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                " has no sip Provider to remove!");
        }

       Iterator listeningPoints=sipStack.getListeningPoints();
        if (listeningPoints!=null) {
            while( listeningPoints.hasNext()) {
                ListeningPoint lp=(ListeningPoint)listeningPoints.next();
                sipStack.deleteListeningPoint(lp);
                listeningPoints=sipStack.getListeningPoints();
                System.out.println("One listening point removed!");
            }
        }
        else {
            ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                " has no listening points to remove!");
        }
        registrar.clean();
    }

    /** Exit the proxy,
     * throws Exception that which can be caught by the upper application
     */
    public void exit()  throws Exception {
        Iterator sipProviders=sipStack.getSipProviders();
        if (sipProviders!=null) {
            while( sipProviders.hasNext()) {
                SipProvider sp=(SipProvider)sipProviders.next();
                sp.removeSipListener(this);
                sipStack.deleteSipProvider(sp);
                sipProviders=sipStack.getSipProviders();
                System.out.println("One sip Provider removed!");
            }
        }
        else {
            ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                " has no sip Provider to remove!");
        }

        Iterator listeningPoints=sipStack.getListeningPoints();
        if (listeningPoints!=null) {
            while( listeningPoints.hasNext()) {
                ListeningPoint lp=(ListeningPoint)listeningPoints.next();
                sipStack.deleteListeningPoint(lp);
                listeningPoints=sipStack.getListeningPoints();
                System.out.println("One listening point removed!");
            }
        }
        else {
            ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                " has no listening points to remove!");
        }
        ProxyDebug.println("Proxy exit........................");
        configuration.listeningPoints.clear();
        registrar.clean();
    }

    public ViaHeader getStackViaHeader() {
        try {
          ListeningPoint lp =
                (ListeningPoint)sipStack.getListeningPoints().next();
          String host = sipStack.getIPAddress();
          int port = lp.getPort();
          String transport = lp.getTransport();
          // branch id is assigned by the transaction layer.
          return  headerFactory.createViaHeader
                        (host,port,transport,null);
        } catch (Exception e) {
                e.printStackTrace();
                return null;
        }
```

```
}

public ContactHeader getStackContactHeader() {
    try {
      ListeningPoint lp =
            (ListeningPoint)sipStack.getListeningPoints().next();
      String host = sipStack.getIPAddress();
      int port = lp.getPort();
      String transport = lp.getTransport();

      SipURI sipURI=addressFactory.createSipURI(null,host);
      sipURI.setPort(port);
      sipURI.setTransportParam(transport);
      Address contactAddress=addressFactory.createAddress(sipURI);

      return headerFactory.createContactHeader(contactAddress);
    } catch (Exception e) {
            e.printStackTrace();
            return null;
    private void callPolling (String caller, String callee){

    }

}


/*************************************************************************/
/************ The main method: to launch the proxy          *************/
/************************************************************************/


public static void main(String args[]) {
    try{
        // the Proxy:
        if (args.length == 0)  {
            System.out.println("Config file missing!");
            System.exit(0);
        }

    private Request transformRequestBasedOnForwardings(Request request) throws ParseException {

            ProxyDebug.println("Proxy [entry]: transformRequestBasedOnForwardings");
            String key=registrar.getKey(request);
            ProxyDebug.println("Request key: " + key);
            ProxyDebug.println("Request: " + request.toString());

            String finalForwardee = registrar.findFinalForwardee(key);
            if (finalForwardee == null) {
                    ProxyDebug.println("Proxy [transformRequestBasedOnForwardings Error]: null finalforwardee");
                    return null;
            }
            ProxyDebug.println("Trying to addressFactory.createSipURI finalForwardee: "+ finalForwardee );

            SipURI finalforwardeeuri = stringToSipURI(finalForwardee);

            ProxyDebug.println("Found final forwardee: "+ finalforwardeeuri );

            Request newrequest = (Request)request.clone();

            ProxyDebug.println("Request header: " + newrequest.getHeader("To"));

            ToHeader tempheader = (ToHeader) newrequest.getHeader(ToHeader.NAME);
            Address newaddressheader = addressFactory.createAddress(finalForwardee);
            tempheader.setAddress(newaddressheader);

        System.out.println("Using configuration file " + args[1]);
        String confFile= (String) args[1];
        Proxy proxy=new Proxy(confFile);
        proxy.start();
        ProxyDebug.println("Proxy ready to work");
    }
    catch(Exception e) {
        System.out.println
        ("ERROR: Set the configuration file flag: " +
        "USE: -cf configuration_file_location.xml"  );
        System.out.println("ERROR, the proxy can not be started, " +
        " exception raised:\n");
        e.printStackTrace();
    }
}

            newrequest.setHeader(tempheader);
            newrequest.setRequestURI(finalforwardeeuri);
            ProxyDebug.println("New Transformed Request: " + newrequest.toString());

            ProxyDebug.println("Proxy [exit]: transformRequestBasedOnForwardings");
            return newrequest;
    }

    // Method for a better URI manipulation with a given string

    public SipURI stringToSipURI(String stringos){
            SipURI newSipURIresult = null;
            try{
                    String[] parts = stringos.split("@");
                    String sipuser = parts[0];
                    String[] sipuserparts = sipuser.split(":");
                    String username = sipuserparts[1];

                    String ipandport = parts[1];
                    String[] ipandportparts = ipandport.split(":");
                    String ipaddress = ipandportparts[0];
                    String portaddress = ipandportparts[1];


                    newSipURIresult = addressFactory.createSipURI(username,ipaddress);
                    newSipURIresult.setPort( Integer.parseInt(portaddress));

            }
            catch(Exception ex){
                    ProxyDebug.println("Error: Couldn't transform String to URI");
            }
            return newSipURIresult;

    }

    /** This is a listener method.
     */
    public void processResponse(ResponseEvent responseEvent) {
            try{

                    //ProxyDebug.println("Pare to stack: "+Thread.currentThread().getStackTrace().toString());

                    for (StackTraceElement ste : Thread.currentThread().getStackTrace()) {
                            ProxyDebug.println(ste.toString());
                    }

                    Response response = responseEvent.getResponse();




                    SipProvider sipProvider = (SipProvider) responseEvent.getSource();
                    ClientTransaction clientTransaction=responseEvent.getClientTransaction();

                    ProxyDebug.println
                    ("\n***************************************************************"+
                                    "\n***************************************************************"+
                                    "\nResponse "+response.getStatusCode() + " "+response.getReasonPhrase()
                                    +" received:\n"+response.toString() );
                    ProxyDebug.println("Processing Response in progress");

                    if (ProxyDebug.debug)
                            ProxyUtilities.printTransaction(clientTransaction);

                    //Henrik - added handling of responses addressed to server
```

```java
                //If we use a presenceserver, and if statuscode was OK...
                CSeqHeader cseqHeader = (CSeqHeader)response.getHeader(CSeqHeader.NAME);

                if (cseqHeader.getMethod().equals("SUBSCRIBE")) {
                        presenceServer.processSubscribeResponse((Response)response.clone(), clientTransaction);
                } else if (cseqHeader.getMethod().equals("NOTIFY")) {
                        //presenceServer.processNotifyResponse((Response)response.clone(), clientTransaction);
                }

                responseForwarding.forwardResponse(sipProvider, response,clientTransaction);

        } catch (Exception ex) {
                if (ProxyDebug.debug) {
                        ProxyDebug.println("Proxy, processResponse(), internal error, "+
                                        "exception raised:");
                        ProxyDebug.logException(ex);
                }
        }
}


/** JAIN Listener method.
 */
public void processTimeout(TimeoutEvent timeOutEvent) {
        ProxyDebug.println("TimeoutEvent received");
        SipProvider sipProvider = (SipProvider)timeOutEvent.getSource();
        TransactionsMapping transactionsMapping = null;
        if (timeOutEvent.isServerTransaction()) {
                ServerTransaction serverTransaction  =
                                timeOutEvent.getServerTransaction();
                Dialog dialog = serverTransaction.getDialog();
                if (dialog != null) {
                        transactionsMapping =
                                        (TransactionsMapping) dialog.getApplicationData();
                        transactionsMapping.removeMapping(serverTransaction);
                }
        } else {
                ClientTransaction clientTransaction =
                                timeOutEvent.getClientTransaction();
                Dialog dialog = clientTransaction.getDialog();
                ServerTransaction st = null;
                if (dialog != null) {
                        transactionsMapping =
                                        (TransactionsMapping) dialog.getApplicationData();
                        if (transactionsMapping != null)  {
                                st = transactionsMapping.getServerTransaction
                                                (clientTransaction);
                        }
                        if (st==null) {
                                ProxyDebug.println
                                ("ERROR, Unable to retrieve the server transaction,"+
                                                " cannot process timeout!");
                                return;
                        }
                } else {
                        ProxyDebug.println
                        ("ERROR, Unable to retrieve the transaction Mapping,"+
                                        " cannot process timeout!");
                        return;
                }
                Request request = st.getRequest();
                // This removes the given mapping from the table but not
                // necessarily the whole thing.
                transactionsMapping.removeMapping(clientTransaction);
                if (!transactionsMapping.hasMapping(st)) {
                        // No more mappings left in the transaction table.
                        try {
                                Response response = messageFactory.createResponse
                                                (Response.REQUEST_TIMEOUT, request);
                                st.sendResponse(response);
                        } catch (ParseException ex) {
                                ex.printStackTrace();
                        } catch (SipException ex1) {
                                ex1.printStackTrace();
                        }
                }
        }


}



/**********************  Methods for         ***************
 *    starting and stopping the proxy                   *
 ************************************************************/

/** Start the proxy, this method has to be called after the init method
 * throws Exception that which can be caught by the upper application
 */
public void start() throws Exception {
        if (configuration!=null
                        && configuration.isValidConfiguration()) {
                Properties properties=new Properties();
                // LOGGING property:

                if (configuration.enableDebug) {
                        ProxyDebug.debug=true;
                        ProxyDebug.setProxyOutputFile(configuration.outputProxy);
                        ProxyDebug.println("DEBUG properties set!");
                        if (configuration.badMessageLogFile!=null)
                                properties.setProperty("gov.nist.javax.sip.BAD_MESSAGE_LOG",
                                                configuration.badMessageLogFile);
                        if (configuration.debugLogFile!=null) {
                                properties.setProperty("gov.nist.javax.sip.DEBUG_LOG",
                                                configuration.debugLogFile);

                        }
                        if (configuration.serverLogFile!=null)
                                properties.setProperty("gov.nist.javax.sip.SERVER_LOG",
                                                configuration.serverLogFile);
                        if (configuration.debugLogFile != null)
                                properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL",
                                                "32");
                        else
                                if (configuration.serverLogFile != null)
                                        properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL",
                                                        "16");
                                else
                                        properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL",
                                                        "0");


                }
                else {
                        System.out.println("DEBUG properties not set!");
                }
                registrar.setExpiresTime(configuration.expiresTime);

                // STOP TIME
                if (configuration.stopTime!=null) {
                        try {
                                long stopTime=Long.parseLong(configuration.stopTime);
                                StopProxy stopProxy=new StopProxy(this);
                                Timer timer=new Timer();
                                timer.schedule(stopProxy,stopTime);
                        }
                        catch(Exception e) {
                                e.printStackTrace();
                        }
                }

                sipStack = null;

                SipFactory sipFactory = SipFactory.getInstance();
                sipFactory.setPathName("gov.nist");
```

```
                        headerFactory = sipFactory.createHeaderFactory();
                        addressFactory = sipFactory.createAddressFactory();
                        messageFactory = sipFactory.createMessageFactory();


                        // Create SipStack object

                        properties.setProperty("javax.sip.IP_ADDRESS",
                                        configuration.stackIPAddress);

                        // We have to add the IP address of the proxy for the domain:
                        configuration.domainList.addElement(configuration.stackIPAddress);
                        ProxyDebug.println("The proxy is responsible for the domain:"+configuration.stackIPAddress);

                        properties.setProperty("javax.sip.STACK_NAME",
                                        configuration.stackName);
                        if (configuration.check(configuration.outboundProxy))
                                properties.setProperty("javax.sip.OUTBOUND_PROXY",
                                                configuration.outboundProxy);
                        if (configuration.check(configuration.routerPath))
                                properties.setProperty("javax.sip.ROUTER_PATH",
                                                configuration.routerPath);
                        if (configuration.check(configuration.extensionMethods))
                                properties.setProperty("javax.sip.EXTENSION_METHODS",
                                                configuration.extensionMethods);
                        // This has to be hardcoded to true. for the proxy.
                        properties.setProperty("javax.sip.RETRANSMISSION_FILTER",
                                        "on");

                        if (configuration.check(configuration.maxConnections) )
                                properties.setProperty("gov.nist.javax.sip.MAX_CONNECTIONS",
                                                configuration.maxConnections);
                        if (configuration.check(configuration.maxServerTransactions) )
                                properties.setProperty("gov.nist.javax.sip.MAX_SERVER_TRANSACTIONS",
                                                configuration.maxServerTransactions);
                        if (configuration.check(configuration.threadPoolSize) )
                                properties.setProperty("gov.nist.javax.sip.THREAD_POOL_SIZE",
                                                configuration.threadPoolSize);

                        if (configuration.domainList!=null)
                                for ( int j=0;j<configuration.domainList.size();j++) {
                                        String domain=(String)configuration.domainList.elementAt(j);
                                        ProxyDebug.println("Here is one domain to take care of:"+domain);
                                }
                        else ProxyDebug.println("No domain to take care of...");

                        if (configuration.accessLogViaRMI) {
                                properties.setProperty("gov.nist.javax.sip.ACCESS_LOG_VIA_RMI",
                                                "true");

                                properties.setProperty("gov.nist.javax.sip.RMI_PORT",
                                                configuration.logRMIPort);

                                if (configuration.check(configuration.logLifetime) )
                                        properties.setProperty("gov.nist.javax.sip.LOG_LIFETIME",
                                                        configuration.logLifetime);

                        }

                        sipStack = sipFactory.createSipStack(properties);



                        // Authentication part:
                        if (configuration.enableAuthentication) {
                                authentication =new Authentication(this);
                                try{

                                        Class authMethodClass =
                                                        Class.forName(configuration.classFile);
                                        AuthenticationMethod authMethod
                                        = (AuthenticationMethod)
                                        authMethodClass.newInstance();
                                        authMethod.initialize(configuration.passwordsFile);

                                        authentication.setAuthenticationMethod(authMethod);

                                }
                                catch(Exception e) {
                                        ProxyDebug.println
                                        ("ERROR, authentication process stopped, exception raised:");
                                        e.printStackTrace();
                                }
                        }

                        // We create the Listening points:
                        Vector lps=configuration.getListeningPoints();

                        for ( int i=0;lps!=null && i<lps.size();i++) {
                                Association a=(Association)lps.elementAt(i);
                                try{
                                        System.out.println("transport  " + a.transport);
                                        System.out.println("port  " +
                                                        Integer.valueOf(a.port).intValue());
                                        ListeningPoint lp=sipStack.createListeningPoint
                                                        (Integer.valueOf(a.port).intValue(),
                                                                        a.transport);
                                        this.listeningPoints.add(lp);
                                        SipProvider sipProvider = sipStack.createSipProvider(lp);
                                        if (this.defaultProvider == null)
                                                this.defaultProvider = sipProvider;
                                        sipProvider.addSipListener( this );
                                }
                                catch(Exception e) {
                                        e.printStackTrace();
                                        ProxyDebug.println
                                        ("ERROR: listening point not created ");
                                }
                        }
                        // Export the registry for polling by the responder.

                        if (configuration.exportRegistry )
                                // initialize the registrar for  RMI.
                                this.registrar.initRMIBindings();


                        // Parse static configuration for registrar.
                        if (configuration.enableRegistrations) {
                                String value=configuration.registrationsFile;
                                ProxyDebug.println("Parsing the XML registrations file: "+value);
                                if (value==null || value.trim().equals("")) {
                                        ProxyDebug.println("You have to set the registrations file...");
                                } else {
                                        registrar.parseXMLregistrations(value);
                                }
                        }
                        else ProxyDebug.println("No registrations to parse...");

                        // Register to proxies if any:
                        registrar.registerToProxies();

                }
                else {
                        System.out.println("ERROR: the configuration file is not correct!"+
                                        " Correct the errors first.");
                }
        }

        /** Stop the proxy, this method has to be called after the start method
         * throws Exception that which can be caught by the upper application
         */
        public void stop()  throws Exception {
                if (sipStack==null) return;
                this.presenceServer.stop();

                Iterator sipProviders=sipStack.getSipProviders();
                if (sipProviders!=null) {
                        while( sipProviders.hasNext()) {
                                SipProvider sp=(SipProvider)sipProviders.next();
                                sp.removeSipListener(this);
                                sipStack.deleteSipProvider(sp);
                                sipProviders=sipStack.getSipProviders();
                                System.out.println("One sip Provider removed!");
                        }
                }
                else {
                        ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                                        " has no sip Provider to remove!");
                }

                Iterator listeningPoints=sipStack.getListeningPoints();
                if (listeningPoints!=null) {
                        while( listeningPoints.hasNext()) {
                                ListeningPoint lp=(ListeningPoint)listeningPoints.next();
                                sipStack.deleteListeningPoint(lp);
                                listeningPoints=sipStack.getListeningPoints();
                                System.out.println("One listening point removed!");
                        }
                }
                else {
                        ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                                        " has no listening points to remove!");
                }
                registrar.clean();
        }

        /** Exit the proxy,
         * throws Exception that which can be caught by the upper application
         */
        public void exit()  throws Exception {
                Iterator sipProviders=sipStack.getSipProviders();
                if (sipProviders!=null) {
                        while( sipProviders.hasNext()) {
                                SipProvider sp=(SipProvider)sipProviders.next();
                                sp.removeSipListener(this);
                                sipStack.deleteSipProvider(sp);
                                sipProviders=sipStack.getSipProviders();
                                System.out.println("One sip Provider removed!");
                        }
                }
                else {
                        ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                                        " has no sip Provider to remove!");
                }

                Iterator listeningPoints=sipStack.getListeningPoints();
                if (listeningPoints!=null) {
                        while( listeningPoints.hasNext()) {
                                ListeningPoint lp=(ListeningPoint)listeningPoints.next();
                                sipStack.deleteListeningPoint(lp);
                                listeningPoints=sipStack.getListeningPoints();
                                System.out.println("One listening point removed!");
                        }
                }
                else {
                        ProxyDebug.println("WARNING, STOP_PROXY, The proxy " +
                                        " has no listening points to remove!");
                }
                ProxyDebug.println("Proxy exit........................");
                configuration.listeningPoints.clear();
                registrar.clean();
        }

        public ViaHeader getStackViaHeader() {
                try {
                        ListeningPoint lp =
                                        (ListeningPoint)sipStack.getListeningPoints().next();
                        String host = sipStack.getIPAddress();
                        int port = lp.getPort();
                        String transport = lp.getTransport();
                        // branch id is assigned by the transaction layer.
                        return  headerFactory.createViaHeader
                                        (host,port,transport,null);
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }

        }

        public ContactHeader getStackContactHeader() {
                try {
                        ListeningPoint lp =
                                        (ListeningPoint)sipStack.getListeningPoints().next();
                        String host = sipStack.getIPAddress();
                        int port = lp.getPort();
                        String transport = lp.getTransport();

                        SipURI sipURI=addressFactory.createSipURI(null,host);
                        sipURI.setPort(port);
                        sipURI.setTransportParam(transport);
                        Address contactAddress=addressFactory.createAddress(sipURI);

                        return headerFactory.createContactHeader(contactAddress);
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }

        }


        /*************************************************************************/
        /************ The main method: to launch the proxy          *************/
        /************************************************************************/


        public static void main(String args[]) {
                try{
                        // the Proxy:
                        if (args.length == 0)  {
                                System.out.println("Config file missing!");
                                System.exit(0);
                        }

                        System.out.println("Using configuration file " + args[1]);
                        String confFile= (String) args[1];
                        Proxy proxy=new Proxy(confFile);
                        proxy.start();
                        ProxyDebug.println("Proxy ready to work");
                }
                catch(Exception e) {
                        System.out.println
                        ("ERROR: Set the configuration file flag: " +
                                        "USE: -cf configuration_file_location.xml"  );
                        System.out.println("ERROR, the proxy can not be started, " +
                                        " exception raised:\n");
                        e.printStackTrace();
                }
        }

}
```

modified: sip-proxy/src/gov/nist/sip/proxy/gui/ListenerProxy.java

```
@ ListenerProxy.java:9 @ import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.rmi.RemoteException;

import gov.nist.sip.proxy.*;
import gov.nist.sip.proxy.Proxy;
import tools.tracesviewer.*;
```

```
@ ListenerProxy.java:25 @ public class ListenerProxy {

    protected Process rmiregistryProcess;
    protected TracesViewer tracesViewer;
    protected String xmlRegistrationsFile;

    public boolean isProxyStarted() {
        return PROXY_STARTED;
    }


    public static void writeFile(String outFile, String text) {
        System.out.println("writeFile: [entry]");
        // we read this file to obtain the options
                try{
                        FileWriter fileWriter = new FileWriter(outFile,false);
                        PrintWriter pw = new PrintWriter(fileWriter,true);
                        //System.out.println(text);
                        if (text==null) {
                                pw.println();
                                //System.out.println("null");
                        }
                        else
                        {
                                pw.println(text);
                                //System.out.println(text);
                        }

                        pw.close();
                        fileWriter.close();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                System.out.println("writeFile: [exit]");
        }

    public void writeXMLRegistrations() {
        System.out.println("writeXMLRegistrations: [entry]");
                String registrationsTags=registrationsTable.getXMLTags();
                //System.out.println(registrationsTags);
                writeFile(xmlRegistrationsFile,registrationsTags);
                System.out.println("writeXMLRegistrations: [exit]");
                System.out.println("OK");
        }

    public ListenerProxy(ProxyLauncher proxyLauncher) {
        this.proxyLauncher=proxyLauncher;
@ ListenerProxy.java:73 @ public class ListenerProxy {
        try{
            configurationFrame=new ConfigurationFrame(proxyLauncher,"Configuration");
            helpBox=new HelpBox();

            xmlRegistrationsFile = proxyLauncher.getProxy().getConfiguration().registrationsFile;
            // First, we have to start a registry for logging the traces
            //startRMIregistry();

@ ListenerProxy.java:83 @ public class ListenerProxy {
        }
    }


        public  RegistrationsTable registrationsTable;

    public void configurationActionPerformed(ActionEvent em){
        try{
@ ListenerProxy.java:181 @ public class ListenerProxy {
        }
    }

    public void statusActionPerformed(ActionEvent ev) {
        try{

                Proxy proxy=proxyLauncher.getProxy();
            Registrar registrar=proxy.getRegistrar();

            if (registrar!=null) {
                ProxyDebug.println("DEBUG, GUI chained to the registrar");
                registrar.setRegistrationsList(proxyLauncher.registrationsList);
            }
            String s = proxyLauncher.registrationsList.getSelectedValue().toString();
            //System.out.println(s);

            registrationsTable=registrar.getRegistrationsTable();
            Iterator iterator=registrationsTable.getRegistrations().keySet().iterator();

            while (iterator!=null && iterator.hasNext()) {
                //ProxyDebug.println("[Registrations Graph here] Vertexes Done!");

                        Registration registration=(Registration)registrationsTable.getRegistrations().get(iterator.next());
                        //ProxyDebug.println("[Registrations Graph Iteration] Got Registration! "+registration.toString());

                        String contact = registration.getKey();
                        //ProxyDebug.println("[Registrations Graph Iteration] Vertex To add! "+vertexToAdd);
                        String[] parts = s.split(" ");
                        //System.out.println(parts[0]);
                        String c = parts[0]+"";
                        System.out.println(c);
                        System.out.println(contact);
                        if(contact.equals(c)){
                                System.out.println(c);
                                if((registration.getuserCategory()).equals("Normal")){
                                        System.out.println("in");
                                        proxyLauncher.registrationsList.updateRegistration(registration, true);
                                        registration.setuserCategory("Premium");
                                        //System.out.println(s);
                                        System.out.println(registration.getuserCategory());
                                        proxyLauncher.registrationsList.updateRegistration(registration, false);
                                        }
                                else{
                                        proxyLauncher.registrationsList.updateRegistration(registration, true);
                                        registration.setuserCategory("Normal");
                                        proxyLauncher.registrationsList.updateRegistration(registration, false);
                                }
                        }
                //System.out.println(registration.getXMLTags());

                }
            this.writeXMLRegistrations();
        }
        catch(Exception e) {
           ProxyDebug.println("ERROR trying to change the status, exception raised:"+e.getMessage());
           //e.printStackTrace();
        }
    }


    public void stopProxy() {
        try{
```

---

modified: sip-proxy/src/gov/nist/sip/proxy/gui/ProxyLauncher.java

---

```
@ ProxyLauncher.java:66 @ public class ProxyLauncher extends JFrame{
    protected JPanel secondPanel;
    protected JPanel thirdPanel;
    protected JPanel fourthPanel;
    protected JPanel fifthPanel;

    protected JButton proxyButton;
    protected JButton statusButton;
    protected JButton traceViewerButton;

    protected RegistrationsList registrationsList;
@ ProxyLauncher.java:150 @ public class ProxyLauncher extends JFrame{
/********************************************************************************/
/********************************************************************************/
/********************************************************************************/
```

```
    public void initComponents() {
        /********************** The main container ****************************/
@ ProxyLauncher.java:288 @ public class ProxyLauncher extends JFrame{
          }
        );

        fifthPanel=new JPanel();
        fifthPanel.setOpaque(false);
        fifthPanel.setBorder(BorderFactory.createEmptyBorder(5,20,10,20));
        container.add(fifthPanel);
        // row, column, gap, gap
        fifthPanel.setLayout( new GridLayout(1,2,5,5) );

        statusButton=new JButton("Change Status: Normal/Premium");
        statusButton.setToolTipText("Please, start/stop the proxy!!!");
        statusButton.setFont(new Font ("Dialog", 1, 14));
        statusButton.setFocusPainted(false);
        statusButton.setBackground(buttonBackGroundColor);
        statusButton.setBorder(buttonBorder);
        statusButton.setVerticalAlignment(AbstractButton.CENTER);
        statusButton.setHorizontalAlignment(AbstractButton.CENTER);
        fifthPanel.add(statusButton);
        statusButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                  listenerProxy.statusActionPerformed(evt);
            }
          }
        );

        traceViewerButton=new JButton("View the traces");
        traceViewerButton.setToolTipText("The traces are waiting for you!!!");
        traceViewerButton.setFont(new Font ("Dialog", 1, 14));
```

modified: sip-proxy/src/gov/nist/sip/proxy/gui/RegistrationsList.java

```
@ RegistrationsList.java:45 @ public class RegistrationsList extends JList {
            if (registrationsTable !=null) {
                Hashtable r=registrationsTable.getRegistrations();
                if (r==null || r.size()==0) {
                    list.addElement("(empty)");
                    return;
                        list.addElement("(empty)");
                        return;
                }
                Enumeration e=r.keys();
                while (e.hasMoreElements()) {
                    String key=(String)e.nextElement();
                    list.addElement(key);
                        String key=(String)e.nextElement();
                        list.addElement(key);
                }
            }
            else list.addElement("(empty)");
@ RegistrationsList.java:63 @ public class RegistrationsList extends JList {
    public  void updateRegistration(Registration registration,boolean toRemove) {

        if (registration!=null ) {
            String  key=registration.getKey(); // key=="sip:user@domain"
            String  key=registration.getKey()+" "+registration.getuserCategory(); // key=="sip:user@domain"
            boolean inList=list.contains(key);
```

modified: sip-proxy/src/gov/nist/sip/proxy/presenceserver/PresenceServer.java

```
@ PresenceServer.java:151 @ public class PresenceServer {
        "\n   Key="+registration.getKey() +
        "\n   DisplayName="+registration.getDisplayName() +
        "\n   Contacts="+registration.getContactsList().toString());

        if(registration.getForwardToUser() != null){
                ProxyDebug.println("\n   Forward To ="+registration.getForwardToUser().toString());
        }
        if(registration.getBlockedUsersList() != null){
                ProxyDebug.println("\n   Blocked User List = " + registration.getBlockedUsersList().toString());
        }

        presentityManager.processRegister(registration.getKey(),Integer.MAX_VALUE);
    }
```

modified: sip-proxy/src/gov/nist/sip/proxy/registrar/Registrar.java

```
@ Registrar.java:22 @ import javax.sip.header.*;
import javax.sip.address.*;
import java.util.*;
import java.net.URLEncoder;
import gov.nist.sip.proxy.Chargement;
import gov.nist.sip.proxy.presenceserver.*;
import gov.nist.sip.proxy.gui.*;
import org.jgraph.JGraph;
import org.jgraph.graph.DefaultEdge;
import org.jgrapht.DirectedGraph;
import org.jgrapht.ListenableGraph;
import org.jgrapht.alg.CycleDetector;
import org.jgrapht.ext.JGraphModelAdapter;
import org.jgrapht.graph.DefaultDirectedGraph;
import org.jgrapht.graph.ListenableDirectedGraph;

//ifdef SIMULATION
/*
import sim.java.net.*;
//endif
*/
 */


/**
@ Registrar.java:54 @ extends UnicastRemoteObject
//
implements RegistrarAccess {

    protected  RegistrationsTable registrationsTable;
    protected  RegistrationsList gui;
    protected  Proxy proxy;

    // in seconds
    public static int EXPIRES_TIME_MIN=1;
    public static int EXPIRES_TIME_MAX=3600;

    protected String xmlRegistrationsFile;
    protected Vector threads;
    /**
     *  Creates new Registrar
     */
    public Registrar(Proxy proxy) throws RemoteException {
        this.proxy = proxy;
        registrationsTable=new RegistrationsTable(this);
    }

    public void registerToProxies() {
        try{
            Configuration configuration=proxy.getConfiguration();
            Vector proxyToRegisterWithList=configuration.proxyToRegisterWithList;
            if (proxyToRegisterWithList!=null) {
                threads=new Vector();
                if (ProxyDebug.debug) {
                    ProxyDebug.println
                    ("Registrar, registerToProxies(), we have to register to "
                    +proxyToRegisterWithList.size()+" proxies");
                }
                for( int i=0;i<proxyToRegisterWithList.size();i++) {
                    Domain domain=
                    (Domain)proxyToRegisterWithList.elementAt(i);
                    if (domain.hostName!=null) {
                        RegistrationDomainThread rr=
                                new RegistrationDomainThread(proxy,domain);
```

```
//ifdef                 SIMULATION
/*
        public  RegistrationsTable registrationsTable;
        protected  RegistrationsList gui;
        protected  Proxy proxy;

        // in seconds
        public static int EXPIRES_TIME_MIN=1;
        public static int EXPIRES_TIME_MAX=3600;

        protected String xmlRegistrationsFile;
        protected Vector threads;
        /**
         *  Creates new Registrar
         */
        public Registrar(Proxy proxy) throws RemoteException {
                this.proxy = proxy;
                registrationsTable=new RegistrationsTable(this);
        }

        public void registerToProxies() {
                try{
                        Configuration configuration=proxy.getConfiguration();
                        Vector proxyToRegisterWithList=configuration.proxyToRegisterWithList;
                        if (proxyToRegisterWithList!=null) {
                                threads=new Vector();
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println
                                        ("Registrar, registerToProxies(), we have to register to "
                                                        +proxyToRegisterWithList.size()+" proxies");
                                }
                                for( int i=0;i<proxyToRegisterWithList.size();i++) {
                                        Domain domain=
                                                        (Domain)proxyToRegisterWithList.elementAt(i);
                                        if (domain.hostName!=null) {
                                                RegistrationDomainThread rr=
                                                                new RegistrationDomainThread(proxy,domain);
                                                //ifdef                 SIMULATION
                                                /*
                        new SimThread(rr).start();
//else
*/
                        new Thread(rr).start();
//endif
//
                        threads.addElement(rr);
                    }
                }
            }
        }
        catch(Exception e) {
            if (ProxyDebug.debug)  {
                ProxyDebug.println
                ("ERROR, Registrar, registerToProxies(), exception  raised:");
            }
            e.printStackTrace();
        }
    }

    public void setRegistrationsList(RegistrationsList registrationsList) {
        this.gui=registrationsList;
    }

    public void parseXMLregistrations(String file) {
        try{
            xmlRegistrationsFile=file;

            XMLRegistrationsParser xmlRegistrationsParser=new
                XMLRegistrationsParser(xmlRegistrationsFile,proxy);
            Registrations registrations=xmlRegistrationsParser.getRegistrations();
            if (registrations==null) return;
            Vector registrationList=registrations.registrationList;
            if (registrationList!=null) {
                if (ProxyDebug.debug) {
                    ProxyDebug.println("Registrar, parseXMLregistrations(), Uploading of "
                                       +registrationList.size()+" registrations");
                }
                for( int i=0;i<registrationList.size();i++) {
                    Registration registration=
                        (Registration)registrationList.elementAt(i);
                    registrationsTable.addRegistration(registration);

                    //Henrik Leion: Add registrations to presenceServer
                    //  (Assuming we are PA for all of them).
                    PresenceServer presenceServer = proxy.getPresenceServer();
                    presenceServer.processUploadedRegistration(registration);

                }
            }


        }
        catch(Exception e) {
           if (ProxyDebug.debug)  {
                ProxyDebug.println
                ("ERROR, Registrar, Registrar(), exception  raised during"+
                " parsing of the static registrations:");
            }
            e.printStackTrace();
        }
    }


    public void clean() {
         if (threads==null) return;
         for( int i=0;i<threads.size();i++) {
                RegistrationDomainThread rr=(RegistrationDomainThread)threads.elementAt(i);
                rr.STOP=true;
         }
    }


     public static void writeFile(String outFile, String text) {
        // we read this file to obtain the options
        try{
            FileWriter fileWriter = new FileWriter(outFile,false);
            PrintWriter pw = new PrintWriter(fileWriter,true);

            if (text==null) {
                pw.println();
            }
            else
            {
                 pw.println(text);
            }

            pw.close();
            fileWriter.close();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

     public void setExpiresTime(int expiresTime) {
        EXPIRES_TIME_MAX=expiresTime;
     }


    public void writeXMLRegistrations() {
        String registrationsTags=registrationsTable.getXMLTags();
        writeFile(xmlRegistrationsFile,registrationsTags);
    }


    public RegistrationsTable getRegistrationsTable() {
        return registrationsTable;
    }
    public Registration getRegistration(String key) {
          return (Registration) registrationsTable.getRegistrations().get(key);
    }


/****************************************************************************/
/************************** RMI REGISTRY    *******************************/

    public String getRegistryXMLTags() throws RemoteException {
        return registrationsTable.getRegistryXMLTags();
    }



    public synchronized Vector getRegistryBindings() throws RemoteException {
        return registrationsTable.getRegistryBindings();
    }

    public synchronized int getRegistrySize() throws RemoteException {
         return registrationsTable.getRegistrySize();
    }

    // Need to add more registry query functions here.
    public void initRMIBindings() {
        String name = null;
        try {
            Configuration configuration=proxy.getConfiguration();
            if (configuration.accessLogViaRMI) {
                SipStack sipStack=proxy.getSipStack();
                Iterator it  =sipStack.getListeningPoints();
                ListeningPoint lp = (ListeningPoint) it.next();
                String stackIPAddress = sipStack.getIPAddress();
                name = "//" + stackIPAddress + ":" + 0 +  "/" +
                sipStack.getStackName()
                + "/" + "test.jainproxy.Registrar";
                if (ProxyDebug.debug)  {
                    ProxyDebug.println("Exporting Registration Table " + name);
                }
                Naming.rebind(name,this);
            }
            else {
               if (ProxyDebug.debug)
                 ProxyDebug.println
                 ("We don't export the registrations because RMI is disabled.");
            }
        }
        catch (Exception ex) {
            if (ProxyDebug.debug) {
                ProxyDebug.println
                 ("Problem trying to export the Registration Table: " + name);
            }
            ex.printStackTrace();
        }
    }

/******************************************************************************/
/******************************************************************************/


    /** Process the register message: add, remove, update the bindings
     *  and manage also the expiration time.
     *  @param Request Register message to set
     *  @return int status code of the process of the Register.
     */
    public
    synchronized void processRegister(Request request, SipProvider sipProvider,
    ServerTransaction serverTransaction ) {
        try{
            MessageFactory messageFactory=proxy.getMessageFactory();

            String key=getKey(request);

            // Add the key if it is a new user:
            if (ProxyDebug.debug){
                ProxyDebug.println
                ("Registrar, processRegister(), key: \""+key+"\"");
            }
            if (key==null){
                if (ProxyDebug.debug) {
                    ProxyDebug.println
                    ("Registrar, processRegister(), key is null"+
                    " 400 INVALID REQUEST replied");
                }
                Response response=messageFactory.createResponse
                (Response.BAD_REQUEST,request);
                if (serverTransaction!=null)
                   serverTransaction.sendResponse(response);
                else sipProvider.sendResponse(response);
                return ;
            }

            // RFC 3261: 10.3:
            /*  6. The registrar checks whether the request contains the Contact
                                                 */
                                                new Thread(rr).start();
                                                //endif
                                                //
                                                threads.addElement(rr);
                                        }
                                }
                        }
                }
                catch(Exception e) {
                        if (ProxyDebug.debug)  {
                                ProxyDebug.println
                                ("ERROR, Registrar, registerToProxies(), exception  raised:");
                        }
                        e.printStackTrace();
                }
        }

        public void setRegistrationsList(RegistrationsList registrationsList) {
                this.gui=registrationsList;
        }

        public void parseXMLregistrations(String file) {
                try{
                        xmlRegistrationsFile=file;

                        XMLRegistrationsParser xmlRegistrationsParser=new
                                        XMLRegistrationsParser(xmlRegistrationsFile,proxy);
                        Registrations registrations=xmlRegistrationsParser.getRegistrations();
                        if (registrations==null) return;
                        Vector registrationList=registrations.registrationList;
                        if (registrationList!=null) {
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println("Registrar, parseXMLregistrations(), Uploading of "
                                                        +registrationList.size()+" registrations");
                                }
                                for( int i=0;i<registrationList.size();i++) {
                                        Registration registration=
                                                        (Registration)registrationList.elementAt(i);
                                        registrationsTable.addRegistration(registration);

                                        //Henrik Leion: Add registrations to presenceServer
                                        //  (Assuming we are PA for all of them).
                                        PresenceServer presenceServer = proxy.getPresenceServer();
                                        presenceServer.processUploadedRegistration(registration);

                                }
                        }
                        registrationsTable.printRegistrations();


                }
                catch(Exception e) {
                        if (ProxyDebug.debug)  {
```

```
                        ProxyDebug.println
                        ("ERROR, Registrar, Registrar(), exception  raised during"+
                                        " parsing of the static registrations:");
                }
                e.printStackTrace();
        }
}


public void clean() {
        if (threads==null) return;
        for( int i=0;i<threads.size();i++) {
                RegistrationDomainThread rr=(RegistrationDomainThread)threads.elementAt(i);
                rr.STOP=true;
        }
}


public static void writeFile(String outFile, String text) {
        // we read this file to obtain the options
        try{
                FileWriter fileWriter = new FileWriter(outFile,false);
                PrintWriter pw = new PrintWriter(fileWriter,true);

                if (text==null) {
                        pw.println();
                }
                else
                {
                        pw.println(text);
                }

                pw.close();
                fileWriter.close();
        }
        catch(Exception e) {
                e.printStackTrace();
        }
}

public static void appendFile(String outFile, String text) {
        // we read this file to obtain the options


        try{
                FileWriter fileWriter = new FileWriter(outFile,true);
                PrintWriter pw = new PrintWriter(fileWriter,true);

                if (text==null) {
                        pw.println();
                }
                else
                {
                        pw.println(text);
                }

                pw.close();
                fileWriter.close();
        }
        catch(Exception e) {
                if (ProxyDebug.debug)  {
                        ProxyDebug.println("Error in append file");
                }
                e.printStackTrace();
        }
}

public void setExpiresTime(int expiresTime) {
        EXPIRES_TIME_MAX=expiresTime;
}


public void writeXMLRegistrations() {
        String registrationsTags=registrationsTable.getXMLTags();
        writeFile(xmlRegistrationsFile,registrationsTags);
}


public RegistrationsTable getRegistrationsTable() {
        return registrationsTable;
}

public Registration getRegistration(String key) {
        return (Registration) registrationsTable.getRegistrations().get(key);
}


/***************************************************************************/
/*************************** RMI REGISTRY    *******************************/

public String getRegistryXMLTags() throws RemoteException {
        return registrationsTable.getRegistryXMLTags();
}



public synchronized Vector getRegistryBindings() throws RemoteException {
        return registrationsTable.getRegistryBindings();
}

public synchronized int getRegistrySize() throws RemoteException {
        return registrationsTable.getRegistrySize();
}

// Need to add more registry query functions here.
public void initRMIBindings() {
        String name = null;
        try {
                Configuration configuration=proxy.getConfiguration();
                if (configuration.accessLogViaRMI) {
                        SipStack sipStack=proxy.getSipStack();
                        Iterator it  =sipStack.getListeningPoints();
                        ListeningPoint lp = (ListeningPoint) it.next();
                        String stackIPAddress = sipStack.getIPAddress();
                        name = "//" + stackIPAddress + ":" + 0 +  "/" +
                                        sipStack.getStackName()
                        + "/" + "test.jainproxy.Registrar";
                        if (ProxyDebug.debug)  {
                                ProxyDebug.println("Exporting Registration Table " + name);
                        }
                        Naming.rebind(name,this);
                }
                else {
                        if (ProxyDebug.debug)
                                ProxyDebug.println
                                ("We don't export the registrations because RMI is disabled.");
                }
        }
        catch (Exception ex) {
                if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Problem trying to export the Registration Table: " + name);
                }
                ex.printStackTrace();
        }
}

/*****************************************************************************/
/*****************************************************************************/


/** Process the register message: add, remove, update the bindings
 *  and manage also the expiration time.
 *  @param Request Register message to set
 *  @return int status code of the process of the Register.
 */
public
synchronized void processRegister(Request request, SipProvider sipProvider,
                        ServerTransaction serverTransaction ) {
                try{
                        MessageFactory messageFactory=proxy.getMessageFactory();

                        String key=getKey(request);

                        // Add the key if it is a new user:
                        if (ProxyDebug.debug){
                                ProxyDebug.println
                                ("Registrar, processRegister(), key: \""+key+"\"");
                        }
                        if (key==null){
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println
                                        ("Registrar, processRegister(), key is null"+
                                                        " 400 INVALID REQUEST replied");
                                }
                                Response response=messageFactory.createResponse
                                                (Response.BAD_REQUEST,request);
                                if (serverTransaction!=null)
                                        serverTransaction.sendResponse(response);
                                else sipProvider.sendResponse(response);
                                return ;
                        }

                        // RFC 3261: 10.3:
                        /*  6. The registrar checks whether the request contains the Contact
         header field.  If not, it skips to the last step.  If the
         Contact header field is present, the registrar checks if there
         is one Contact field value that contains the special value "*"
@ Registrar.java:334 @ implements RegistrarAccess {
         remove the binding only if the CSeq in the request is higher
         than the value stored for that binding.  Otherwise, the update
         MUST be aborted and the request fails.
            */

            if ( !hasContactHeaders(request) ) {
                Vector contactHeaders=getContactHeaders(key);
                Response response=messageFactory.createResponse
                (Response.OK,request);
                if ( contactHeaders!=null ) {
                    for (int i = 0 ; i < contactHeaders.size(); i++) {
                        ContactHeader contact = (ContactHeader)
                        contactHeaders.elementAt(i);
                        response.addHeader(contact);
                    }
                }

                if (serverTransaction!=null)
                    serverTransaction.sendResponse(response);
                else sipProvider.sendResponse(response);
                if (ProxyDebug.debug) {
                    ProxyDebug.println
                    ("Registrar, processRegister(), response sent:"+response.toString());
                }
                return;
            }


            // bug report by Alistair Coles
            if ( hasStar(request) ) {
                Vector contactHeaders=getContactHeaders(key);
                if (contactHeaders.size()>1) {
                    if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), more than one contact header"+
                        " is present at the same time as a wild card."+
                        " 400 INVALID REQUEST replied");
                    }
                    Response response=messageFactory.createResponse
                    (Response.BAD_REQUEST,request);
                     if (serverTransaction!=null)
                        serverTransaction.sendResponse(response);
                    else sipProvider.sendResponse(response);
                    if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), response sent:");
                        ProxyDebug.print(response.toString());
                    }
                    return ;
                }

                if ( !hasExpiresZero(request) ) {
                    if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), expires time different from"+
                        " 0 with a wild card."+
                        " 400 INVALID REQUEST replied");
                    }
                    Response response=messageFactory.createResponse
                    (Response.BAD_REQUEST,request);

                    if (serverTransaction!=null)
                        serverTransaction.sendResponse(response);
                    else sipProvider.sendResponse(response);
                    if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), response sent:");
                        ProxyDebug.print(response.toString());
                    }
                    return ;
                }

                if (ProxyDebug.debug) {
                    ProxyDebug.println
                    ("Registrar, processRegister(), (* and expires=0) "+
                    " we remove the registration!!");
                }
                registrationsTable.removeRegistration(key);

                Response response=messageFactory.createResponse
                (Response.OK,request);

                if (serverTransaction!=null)
                    serverTransaction.sendResponse(response);
                else sipProvider.sendResponse(response);

                if (ProxyDebug.debug) {
                    ProxyDebug.println
                    ("Registrar, processRegister(), response sent:");
                    ProxyDebug.print(response.toString());
                }
                return;
            }


            if ( registrationsTable.hasRegistration(key) ) {
                registrationsTable.updateRegistration(key,request);

                if (  proxy.getConfiguration().rfc2543Compatible &&
                key.indexOf(":5060") < 0 ) {
                    //
                    // Hack for Cisco IP Phone which registers incorrectly
                    // by not specifying :5060.
                    //
                    key += ":5060";

                    System.out.println("CISCO IP PHONE FIX:  "
                        + "Updating proper registration for "
                        + key);

                    registrationsTable.updateRegistration(key, request);
                }

                Vector contactHeaders=getContactHeaders(key);
                Response response=
                messageFactory.createResponse(Response.OK,request);
                         */
```

```
if ( !hasContactHeaders(request) ) {
        Vector contactHeaders=getContactHeaders(key);
        Response response=messageFactory.createResponse
                        (Response.OK,request);
        if ( contactHeaders!=null ) {
                for (int i = 0 ; i < contactHeaders.size(); i++) {
                        ContactHeader contact = (ContactHeader)
                                        contactHeaders.elementAt(i);
                        response.addHeader(contact);
                }
        }

        if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
        else sipProvider.sendResponse(response);
        if (ProxyDebug.debug) {
                ProxyDebug.println
                ("Registrar, processRegister(), response sent:"+response.toString());
        }

        // if registrations table has changed we should update the registrations.xml file
        this.writeXMLRegistrations();

        return;
}


// bug report by Alistair Coles
if ( hasStar(request) ) {
        Vector contactHeaders=getContactHeaders(key);
        if (contactHeaders.size()>1) {
                if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), more than one contact header"+
                                        " is present at the same time as a wild card."+
                                        " 400 INVALID REQUEST replied");
                }
                Response response=messageFactory.createResponse
                                (Response.BAD_REQUEST,request);
                if (serverTransaction!=null)
                        serverTransaction.sendResponse(response);
                else sipProvider.sendResponse(response);
                if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), response sent:");
                        ProxyDebug.print(response.toString());
                }
                return ;
        }

        if ( !hasExpiresZero(request) ) {
                if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), expires time different from"+
                                        " 0 with a wild card."+
                                        " 400 INVALID REQUEST replied");
                }
                Response response=messageFactory.createResponse
                                (Response.BAD_REQUEST,request);

                if (serverTransaction!=null)
                        serverTransaction.sendResponse(response);
                else sipProvider.sendResponse(response);
                if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), response sent:");
                        ProxyDebug.print(response.toString());
                }
                return ;
        }

        if (ProxyDebug.debug) {
                ProxyDebug.println
                ("Registrar, processRegister(), (* and expires=0) "+
                                " we remove the registration!!");
        }
        registrationsTable.removeRegistration(key);

        Response response=messageFactory.createResponse
                        (Response.OK,request);

        if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
        else sipProvider.sendResponse(response);

        if (ProxyDebug.debug) {
                ProxyDebug.println
                ("Registrar, processRegister(), response sent:");
                ProxyDebug.print(response.toString());
        }
        return;
}



/*
 * 29-1-2017
 * Check if the username and password matches to a current registration
 * If he username is the same and the passsword is not equal then we send
 * BAD Responese to the sender
 */

String content = new String( request.getRawContent());
String pass = content;
String username = getKey(request);

ProxyDebug.println("User: "+username+" Content of request: "+pass + " me tou kwsta ta skoulikia: " + content);


try{
        pass = content.split("Password:")[1];
        pass = pass.replace("\n","");

}
catch(Exception e){
        e.printStackTrace();
        ProxyDebug.println("Could not get proper password for User: "+username);
}

ProxyDebug.println("User: "+username+" trying to connect with password: "+pass);


if ( registrationsTable.hasRegistration(key) ) {


        Hashtable registrations = registrationsTable.getRegistrations();
        Registration registration=(Registration) registrations.get(username);

        ProxyDebug.println("Given Password: "+pass+" Saved Password: "+registration.getPassword());
        if (!registration.getPassword().equals(pass)){

                // username same and password not the same :(

                if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), username same and password not the same!");
                }
                Response response=messageFactory.createResponse
                                (Response.UNAUTHORIZED,request);
                if (serverTransaction!=null)
                        serverTransaction.sendResponse(response);
                else sipProvider.sendResponse(response);
                if (ProxyDebug.debug) {
                        ProxyDebug.println
                        ("Registrar, processRegister(), UNAUTHORIZED response sent:");
                        ProxyDebug.print(response.toString());
```

```
				}
				return ;

			}



			registrationsTable.updateRegistration(key,request);

			// if registrations table has changed we should update the registrations.xml file
			this.writeXMLRegistrations();

			if (  proxy.getConfiguration().rfc2543Compatible &&
					key.indexOf(":5060") < 0 ) {
				//
				// Hack for Cisco IP Phone which registers incorrectly
				// by not specifying :5060.
				//
				key += ":5060";

				System.out.println("CISCO IP PHONE FIX:  "
						+ "Updating proper registration for "
						+ key);

				registrationsTable.updateRegistration(key, request);
			}

			Vector contactHeaders=getContactHeaders(key);
			Response response=
					messageFactory.createResponse(Response.OK,request);
			try{
				if ( hasExpiresZero(request) ) {
					response.addHeader(request.getHeader(ExpiresHeader.NAME));
				}
			}
			catch(Exception e){
				e.printStackTrace();
			}
			if ( contactHeaders!=null ) {
				for (int i = 0; i < contactHeaders.size(); i++) {
					ContactHeader contact = (ContactHeader)
							contactHeaders.elementAt(i);
					response.addHeader(contact);
				}
			}

			if (serverTransaction!=null)
				serverTransaction.sendResponse(response);
			else sipProvider.sendResponse(response);

			if (ProxyDebug.debug)  {
				ProxyDebug.println
				("Registrar, processRegister(), response sent:");
				ProxyDebug.print(response.toString());
			}
		}
		else {
			// Let's check the Expires header:
			if ( hasExpiresZero(request) ) {
				// This message is lost....
				proxy.getPresenceServer();
				ProxyDebug.println("Registrar, processRegister(), "+
						"we don't have any record for this REGISTER.");
				Response response=messageFactory.createResponse
						(Response.OK,request);

				if (serverTransaction!=null)
					serverTransaction.sendResponse(response);
				else sipProvider.sendResponse(response);
				if (ProxyDebug.debug) {
					ProxyDebug.println
					("Registrar, processRegister(), response sent:");
					ProxyDebug.print(response.toString());
				}
				return;
			}

			registrationsTable.addRegistration(key,request);

			// if registrations table has changed we should update the registrations.xml file
			this.writeXMLRegistrations();

			if (proxy.getConfiguration().rfc2543Compatible &&
					key.indexOf(":5060") < 0) {
				//
				// Hack for Cisco IP Phone which registers incorrectly
				// by not specifying :5060.
				//
				key += ":5060";

				System.out.println("CISCO IP PHONE FIX:  "
						+ "adding proper registration for " + key);

				registrationsTable.addRegistration(key, request);

				// if registrations table has changed we should update the registrations.xml file
				this.writeXMLRegistrations();
			}

			// we have to forward SUBSCRIBE if the presence server
			// is enabled:

			if (proxy.isPresenceServer()) {
				PresenceServer presenceServer=
						proxy.getPresenceServer();
				ProxyDebug.println("Registrar, processRegister(), "+
						"  we have to check if we have some SUBSCRIBE stored.");
			}

			Vector contactHeaders=getContactHeaders(key);
			Response response=
					messageFactory.createResponse(Response.OK,request);
			if ( contactHeaders!=null ) {
				for (int i = 0; i < contactHeaders.size(); i++) {
					ContactHeader contact = (ContactHeader)
							contactHeaders.elementAt(i);
					response.addHeader(contact);
				}
			}

			if (serverTransaction!=null)
				serverTransaction.sendResponse(response);
			else sipProvider.sendResponse(response);

			if (ProxyDebug.debug)  {
				ProxyDebug.println
				("Registrar, processRegister(), response sent:");
				ProxyDebug.print(response.toString());
			}
		}
	}  catch (IOException ex) {
		if (ProxyDebug.debug) {
			ProxyDebug.println("Registrar exception raised:");
			ProxyDebug.logException(ex);
		}
	} catch (SipException ex) {
		if (ProxyDebug.debug) {
			ProxyDebug.println("Registrar exception raised:");
			ProxyDebug.logException(ex);
		}
	} catch(Exception ex) {
		if (ProxyDebug.debug) {
			ProxyDebug.println
			("Registrar, processRegister(), internal error, "+
					"exception raised:");
			ProxyDebug.logException(ex);
		}
```

```
            }
    }

    public static URI getCleanUri(URI uri) {
            if (uri instanceof SipURI) {
                    SipURI sipURI=(SipURI)uri.clone();

                    Iterator iterator=sipURI.getParameterNames();
                    while (iterator!=null && iterator.hasNext()) {
                            String name=(String)iterator.next();
                            sipURI.removeParameter(name);
                    }
                    return  sipURI;
            }
            else return  uri;
    }

    /** The key is built following this rule:
     * The registrar extracts the address-of-record from the To header
     * field of the request. The URI
     * MUST then be converted to a canonical form.  To do that, all
     * URI parameters MUST be removed (including the user-param), and
     * any escaped characters MUST be converted to their unescaped
     * form.  The result serves as an index into the list of bindings
     */
    public String getKey(Request request) {
            // Let's see if we already have a binding for this request:
            try{
                if ( hasExpiresZero(request) ) {
                     response.addHeader(request.getHeader(ExpiresHeader.NAME));
                }
            }
            catch(Exception e){
                e.printStackTrace();
            }
            if ( contactHeaders!=null ) {
                for (int i = 0; i < contactHeaders.size(); i++) {
                    ContactHeader contact = (ContactHeader)
                    contactHeaders.elementAt(i);
                    response.addHeader(contact);
                }
            }

             if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
            else sipProvider.sendResponse(response);

            if (ProxyDebug.debug)  {
                ProxyDebug.println
                ("Registrar, processRegister(), response sent:");
                ProxyDebug.print(response.toString());
            }
        }
        else {
            // Let's check the Expires header:
            if ( hasExpiresZero(request) ) {
                // This message is lost....
                proxy.getPresenceServer();
                ProxyDebug.println("Registrar, processRegister(), "+
                "we don't have any record for this REGISTER.");
                Response response=messageFactory.createResponse
                (Response.OK,request);

                if (serverTransaction!=null)
                    serverTransaction.sendResponse(response);
                else sipProvider.sendResponse(response);
                if (ProxyDebug.debug) {
                    ProxyDebug.println
                    ("Registrar, processRegister(), response sent:");
                    ProxyDebug.print(response.toString());
                }
                return;
            }

            registrationsTable.addRegistration(key,request);

            if (proxy.getConfiguration().rfc2543Compatible &&
                key.indexOf(":5060") < 0) {
                //
                // Hack for Cisco IP Phone which registers incorrectly
                // by not specifying :5060.
                //
                key += ":5060";

                System.out.println("CISCO IP PHONE FIX:  "
                    + "adding proper registration for " + key);

                registrationsTable.addRegistration(key, request);
            }

            // we have to forward SUBSCRIBE if the presence server
            // is enabled:

            if (proxy.isPresenceServer()) {
                PresenceServer presenceServer=
                proxy.getPresenceServer();
                ProxyDebug.println("Registrar, processRegister(), "+
                "  we have to check if we have some SUBSCRIBE stored.");
            }

            Vector contactHeaders=getContactHeaders(key);
            Response response=
            messageFactory.createResponse(Response.OK,request);
            if ( contactHeaders!=null ) {
                for (int i = 0; i < contactHeaders.size(); i++) {
                    ContactHeader contact = (ContactHeader)
                    contactHeaders.elementAt(i);
                    response.addHeader(contact);
                }
            }

             if (serverTransaction!=null)
                serverTransaction.sendResponse(response);
             else sipProvider.sendResponse(response);

            if (ProxyDebug.debug)  {
                ProxyDebug.println
                ("Registrar, processRegister(), response sent:");
                ProxyDebug.print(response.toString());
            }
        }
    }  catch (IOException ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println("Registrar exception raised:");
            ProxyDebug.logException(ex);
         }
    } catch (SipException ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println("Registrar exception raised:");
            ProxyDebug.logException(ex);
         }
    } catch(Exception ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println
            ("Registrar, processRegister(), internal error, "+
            "exception raised:");
            ProxyDebug.logException(ex);
        }
    }
}

public static URI getCleanUri(URI uri) {
    if (uri instanceof SipURI) {
        SipURI sipURI=(SipURI)uri.clone();

        Iterator iterator=sipURI.getParameterNames();
        while (iterator!=null && iterator.hasNext()) {
            String name=(String)iterator.next();
            sipURI.removeParameter(name);
        }
        return  sipURI;
    }
    else return  uri;
}

/** The key is built following this rule:
 * The registrar extracts the address-of-record from the To header
 * field of the request. The URI
 * MUST then be converted to a canonical form.  To do that, all
 * URI parameters MUST be removed (including the user-param), and
 * any escaped characters MUST be converted to their unescaped
 * form.  The result serves as an index into the list of bindings
 */
public String getKey(Request request) {
    // Let's see if we already have a binding for this request:
    try{
        ToHeader toHeader=(ToHeader)request.getHeader(ToHeader.NAME);
        Address address=null;
        address  = toHeader.getAddress();

        javax.sip.address.URI  cleanedUri;
        if (address==null) {
            cleanedUri= getCleanUri(request.getRequestURI());
        }
        else {
            // We have to build the key, all
            // URI parameters MUST be removed:
            cleanedUri = getCleanUri(address.getURI());
        }
        String  keyresult=cleanedUri.toString();

        return keyresult.toLowerCase();
    } catch(Exception ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println("Registrar, hasDomainRegistered(), internal error, "+
            "exception raised:");
            ProxyDebug.logException(ex);
        }
        return null;
    }
}

public boolean hasRegistration(String key) {
    return registrationsTable.hasRegistration(key);
}

public boolean hasDomainRegistered(Request request) {
    try{
        URI uri=request.getRequestURI();
        URI cleanedURI=getCleanUri(uri);

        if (! (cleanedURI instanceof SipURI) ) return false;

        // We have to check the host part:
        String host=((SipURI)cleanedURI).getHost();

        return hasRegistration("sip:"+host );
    }
    catch (Exception ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println("Registrar, hasDomainRegistered(), internal error, "+
            "exception raised:");
            ProxyDebug.logException(ex);
        }
        return false;
    }
}


public boolean hasDomainRegistered(URI uri) {
    try{
        URI cleanedURI=getCleanUri(uri);

        if (! (cleanedURI instanceof SipURI) ) return false;

        // We have to check the host part:
        String host=((SipURI)cleanedURI).getHost();

        return hasRegistration("sip:"+host );
    }
    catch (Exception ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println("Registrar, hasDomainRegistered(), internal error, "+
            "exception raised:");
            ProxyDebug.logException(ex);
        }
        return false;
    }

}


public Vector getDomainContactsURI(Request request) {
    try{
        URI uri=request.getRequestURI();
        URI cleanedURI=getCleanUri(uri);

        if (! (cleanedURI instanceof SipURI) ) return null;

        // We have to check the host part:
        String host=((SipURI)cleanedURI).getHost();

        Vector contacts=getContactHeaders("sip:"+ host );
        if (contacts==null) return null;
        Vector results=new Vector();
        for (int i=0;i<contacts.size();i++) {
            ContactHeader contact = (ContactHeader)
                contacts.elementAt(i);
            Address address=contact.getAddress();
            uri=address.getURI();
            cleanedURI=getCleanUri(uri);
            results.addElement(cleanedURI);
        }
        return results;
    }
    catch (Exception ex) {
       if (ProxyDebug.debug) {
            ProxyDebug.println("Registrar, getDomainContacts(), internal error, "+
            "exception raised:");
            ProxyDebug.logException(ex);
        }
        return null;
    }
}


public boolean hasRegistration(Request request)   {
    try{
        String key = getKey(request);
        return hasRegistration(key);
    }
    catch (Exception ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println("Registrar, hasRegistration(), internal error, "+
            "exception raised:");
            ProxyDebug.logException(ex);
        }
        return false;
    }
}
 /*
  * The result is a list of URI that we kept from a registration related
  * to the ToHeader URI from this request.
  */
public Vector getContactsURI(Request request) {
    try{
        String key=getKey(request);
        Vector contacts=getContactHeaders(key);
        if (contacts==null) return null;
        Vector results=new Vector();
        for (int i=0;i<contacts.size();i++) {
            ContactHeader contact = (ContactHeader)
                contacts.elementAt(i);
            Address address=contact.getAddress();
            URI uri=address.getURI();
            URI cleanedURI=getCleanUri(uri);
            results.addElement(cleanedURI);
        }
        return results;
    }
    catch (Exception ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println
            ("Registrar, getContactsURI(), internal error, exception raised:");
            ProxyDebug.logException(ex);
        }
        return null;
    }
}
/*
  * Matches a Sip URI "sip:user@domain" with a list of Contacts
  * @param key The sip URI found in the To-header of a request
  * @author Henrik Leion
  */
public Vector getContactsURI(String key) {
    try{
        Vector contacts=getContactHeaders(key);
        if (contacts==null) return null;
        Vector results=new Vector();
        for (int i=0;i<contacts.size();i++) {
            ContactHeader contact = (ContactHeader)
                contacts.elementAt(i);
            Address address=contact.getAddress();
            URI uri=address.getURI();
            URI cleanedURI=getCleanUri(uri);
            results.addElement(cleanedURI);
        }
        return results;
    }
    catch (Exception ex) {
        if (ProxyDebug.debug) {
            ProxyDebug.println
            ("Registrar, getContactsURI(), internal error, exception raised:");
            ProxyDebug.logException(ex);
        }
        return null;
    }
}

public boolean hasContactHeaders(Request request) {
     ListIterator list=(ListIterator)request.getHeaders(ContactHeader.NAME);
     return list!=null;
}

private boolean hasStar(Request request) throws Exception{
    ListIterator list=(ListIterator)request.getHeaders(ContactHeader.NAME);

    if (list==null) return false;
    while( list.hasNext() ) {
         ContactHeader contactHeader=(ContactHeader)list.next();
         if (contactHeader.getAddress().isWildcard()  ) {
             return true;
         }
    }
    return false;
}

private boolean hasExpiresZero(Request request) {
    try{
        ExpiresHeader expiresHeader=
            (ExpiresHeader)request.getHeader(ExpiresHeader.NAME);
        if (expiresHeader==null) {
              ProxyDebug.println
            ("Registrar, hasExpiresZero(), the REGISTER does not have an Expires Header");
            return false;
        }
        else
        {
              ProxyDebug.println
            ("Registrar, hasExpiresZero(), the REGISTER has an Expires Header with"+
            " expires time:" +expiresHeader.getExpires());
            return expiresHeader.getExpires()==0;
        }
    }
    catch(Exception e){
        if (ProxyDebug.debug) {
            ProxyDebug.println
            ("Registrar, hasExpiresZero(), internal error, exception raised:");
            ProxyDebug.logException(e);
        }
        return false;
    }
}


public Vector getContactHeaders(String key) {
    return registrationsTable.getContactHeaders(key);
}

public static Vector getContactHeaders(Request request){
    Vector contacts =new Vector();
    try{
        ListIterator list=
            (ListIterator)request.getHeaders(ContactHeader.NAME);

        if (list==null) return contacts;
        while( list.hasNext() ) {
            ContactHeader contactHeader=(ContactHeader)list.next();
            contacts.addElement(contactHeader);
        }

        // We will sort out the contacts following the "q" parameter

        return contacts;
    }
    catch(Exception e){
        if (ProxyDebug.debug) {
            ProxyDebug.println
            ("Registrar, getContactHeaders(), internal error, exception raised:");
            ProxyDebug.logException(e);
        }
        return contacts;
    }
}

protected void printRegistrations(){
    registrationsTable.printRegistrations();
}

                    ToHeader toHeader=(ToHeader)request.getHeader(ToHeader.NAME);
                    Address address=null;
                    address  = toHeader.getAddress();

                    javax.sip.address.URI  cleanedUri;
                    if (address==null) {
                            cleanedUri= getCleanUri(request.getRequestURI());
                    }
                    else {
                            // We have to build the key, all
                            // URI parameters MUST be removed:
                            cleanedUri = getCleanUri(address.getURI());
                    }
                    String  keyresult=cleanedUri.toString();
                    return keyresult.toLowerCase();
            } catch(Exception ex) {
                    if (ProxyDebug.debug) {
                            ProxyDebug.println("Registrar, hasDomainRegistered(), internal error, "+
                                            "exception raised:");
                            ProxyDebug.logException(ex);
                    }
                    return null;
            }
    }

    public boolean hasRegistration(String key) {
            return registrationsTable.hasRegistration(key);
    }

    public boolean hasDomainRegistered(Request request) {
            try{
                    URI uri=request.getRequestURI();
                    URI cleanedURI=getCleanUri(uri);

                    if (! (cleanedURI instanceof SipURI) ) return false;

                    // We have to check the host part:
                            String host=((SipURI)cleanedURI).getHost();

                            return hasRegistration("sip:"+host );
            }
            catch (Exception ex) {
                    if (ProxyDebug.debug) {
                            ProxyDebug.println("Registrar, hasDomainRegistered(), internal error, "+
                                            "exception raised:");
                            ProxyDebug.logException(ex);
                    }
                    return false;
            }
    }


    public boolean hasDomainRegistered(URI uri) {
            try{
                    URI cleanedURI=getCleanUri(uri);

                    if (! (cleanedURI instanceof SipURI) ) return false;

                    // We have to check the host part:
                            String host=((SipURI)cleanedURI).getHost();

                            return hasRegistration("sip:"+host );
            }
            catch (Exception ex) {
                    if (ProxyDebug.debug) {
                            ProxyDebug.println("Registrar, hasDomainRegistered(), internal error, "+
                                            "exception raised:");
                            ProxyDebug.logException(ex);
                    }
                    return false;
            }

    }


    public Vector getDomainContactsURI(Request request) {
            try{
                    URI uri=request.getRequestURI();
                    URI cleanedURI=getCleanUri(uri);

                    if (! (cleanedURI instanceof SipURI) ) return null;

                    // We have to check the host part:
                            String host=((SipURI)cleanedURI).getHost();

                            Vector contacts=getContactHeaders("sip:"+ host );
                            if (contacts==null) return null;
                            Vector results=new Vector();
                            for (int i=0;i<contacts.size();i++) {
                                    ContactHeader contact = (ContactHeader)
                                                    contacts.elementAt(i);
                                    Address address=contact.getAddress();
                                    uri=address.getURI();
                                    cleanedURI=getCleanUri(uri);
                                    results.addElement(cleanedURI);
                            }
                            return results;
            }
            catch (Exception ex) {
                    if (ProxyDebug.debug) {
                            ProxyDebug.println("Registrar, getDomainContacts(), internal error, "+
                                            "exception raised:");
                            ProxyDebug.logException(ex);
                    }
                    return null;
            }
    }


    public boolean hasRegistration(Request request)   {
            try{
                    String key = getKey(request);
                    return hasRegistration(key);
            }
            catch (Exception ex) {
                    if (ProxyDebug.debug) {
                            ProxyDebug.println("Registrar, hasRegistration(), internal error, "+
                                            "exception raised:");
                            ProxyDebug.logException(ex);
                    }
                    return false;
            }
    }
    /*
     * The result is a list of URI that we kept from a registration related
     * to the ToHeader URI from this request.
     */
    public Vector getContactsURI(Request request) {
            try{
                    String key=getKey(request);
                    Vector contacts=getContactHeaders(key);
                    if (contacts==null) return null;
                    Vector results=new Vector();
                    for (int i=0;i<contacts.size();i++) {
                            ContactHeader contact = (ContactHeader)
                                            contacts.elementAt(i);
                            Address address=contact.getAddress();
                            URI uri=address.getURI();
                            URI cleanedURI=getCleanUri(uri);
                            results.addElement(cleanedURI);
                    }
                    return results;
            }
            catch (Exception ex) {
                    if (ProxyDebug.debug) {
                            ProxyDebug.println
                            ("Registrar, getContactsURI(), internal error, exception raised:");
                            ProxyDebug.logException(ex);
                    }
                    return null;
            }
    }
    /*
     * Matches a Sip URI "sip:user@domain" with a list of Contacts
     * @param key The sip URI found in the To-header of a request
     * @author Henrik Leion
     */
    public Vector getContactsURI(String key) {
            try{
                    Vector contacts=getContactHeaders(key);
                    if (contacts==null) return null;
                    Vector results=new Vector();
```

```java
                        for (int i=0;i<contacts.size();i++) {
                                ContactHeader contact = (ContactHeader)
                                                contacts.elementAt(i);
                                Address address=contact.getAddress();
                                URI uri=address.getURI();
                                URI cleanedURI=getCleanUri(uri);
                                results.addElement(cleanedURI);
                        }
                        return results;
                }
                catch (Exception ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, getContactsURI(), internal error, exception raised:");
                                ProxyDebug.logException(ex);
                        }
                        return null;
                }
        }

        public boolean hasContactHeaders(Request request) {
                ListIterator list=(ListIterator)request.getHeaders(ContactHeader.NAME);
                return list!=null;
        }

        private boolean hasStar(Request request) throws Exception{
                ListIterator list=(ListIterator)request.getHeaders(ContactHeader.NAME);

                if (list==null) return false;
                while( list.hasNext() ) {
                        ContactHeader contactHeader=(ContactHeader)list.next();
                        if (contactHeader.getAddress().isWildcard()  ) {
                                return true;
                        }
                }
                return false;
        }

        private boolean hasExpiresZero(Request request) {
                try{
                        ExpiresHeader expiresHeader=
                                        (ExpiresHeader)request.getHeader(ExpiresHeader.NAME);
                        if (expiresHeader==null) {
                                ProxyDebug.println
                                ("Registrar, hasExpiresZero(), the REGISTER does not have an Expires Header");
                                return false;
                        }
                        else
                        {
                                ProxyDebug.println
                                ("Registrar, hasExpiresZero(), the REGISTER has an Expires Header with"+
                                                " expires time:" +expiresHeader.getExpires());
                                return expiresHeader.getExpires()==0;
                        }
                }
                catch(Exception e){
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, hasExpiresZero(), internal error, exception raised:");
                                ProxyDebug.logException(e);
                        }
                        return false;
                }
        }


        public Vector getContactHeaders(String key) {
                return registrationsTable.getContactHeaders(key);
        }

        public static Vector getContactHeaders(Request request){
                Vector contacts =new Vector();
                try{
                        ListIterator list=
                                        (ListIterator)request.getHeaders(ContactHeader.NAME);

                        if (list==null) return contacts;
                        while( list.hasNext() ) {
                                ContactHeader contactHeader=(ContactHeader)list.next();
                                contacts.addElement(contactHeader);
                        }

                        // We will sort out the contacts following the "q" parameter

                        return contacts;
                }
                catch(Exception e){
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, getContactHeaders(), internal error, exception raised:");
                                ProxyDebug.logException(e);
                        }
                        return contacts;
                }
        }

        protected void printRegistrations(){
                registrationsTable.printRegistrations();
        }



        /*
         * Add the response to the user who wants to perform forwarding 22-1-2017
         */
        public synchronized void processUserInfo(Request request, SipProvider sipProvider,
                        ServerTransaction serverTransaction, HeaderFactory headerFactory ) {
                try{
                        MessageFactory messageFactory=proxy.getMessageFactory();

                        String key=getKey(request);

                        // Add the key if it is a new user:
                        if (ProxyDebug.debug){
                                ProxyDebug.println
                                ("Registrar, processUserInfo(), key: \""+key+"\"");
                        }
                        if (key==null){
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println
                                        ("Registrar, processUserInfo(), key is null"+
                                                        " 400 INVALID REQUEST replied");
                                }
                                Response response=messageFactory.createResponse
                                                (Response.BAD_REQUEST,request);
                                if (serverTransaction!=null)
                                        serverTransaction.sendResponse(response);
                                else sipProvider.sendResponse(response);
                                return ;
                        }


                        if ( registrationsTable.hasRegistration(key) ) {
                                        Hashtable registrations=registrationsTable.getRegistrations();
                                Registration registration=(Registration)registrations.get(key);

                                        //if we are here everything went as planned
                                        Response response=
                                                        messageFactory.createResponse(Response.OK,request);

                                        ContentTypeHeader hbill = headerFactory.createContentTypeHeader("application", "info");

                                        String textToSend = "";
                                        if(registration.getForwardToUser()!= null)
                                                textToSend = "Forward:"+registration.getForwardToUser()+"\n";
                                        else
                                                textToSend = "Forward:\n";
                                        Vector BlockedUsersList = registration.getBlockedUsersList();
                                        if (BlockedUsersList != null){
                                        Iterator itr = BlockedUsersList.iterator();
                                        while (itr.hasNext()){
                                                String CurrentBlockedUser = itr.next().toString();
                                                textToSend = textToSend + "BlockedUser:"+CurrentBlockedUser+"\n";

                                        }
                                }

                                        response.setContent(textToSend, hbill);

                                        if (serverTransaction!=null)
                                                serverTransaction.sendResponse(response);
                                        else sipProvider.sendResponse(response);

                                        if (ProxyDebug.debug)  {
                                                ProxyDebug.println
                                                ("Registrar, processUserInfo(), response sent:");
                                                ProxyDebug.print(response.toString());
                                        }

                                        return;

                        }


                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("No valid User to send him INFO");
                        }
                        Response response=messageFactory.createResponse
                                        (Response.BAD_REQUEST,request);
                        if (serverTransaction!=null)
                                serverTransaction.sendResponse(response);
                        else sipProvider.sendResponse(response);
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserInfo(), response sent:");
                                ProxyDebug.print(response.toString());
                        }
                        return ;


                } catch (SipException ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println("Registrar.processUserInfo exception raised:");
                                ProxyDebug.logException(ex);
                        }
                } catch(Exception ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserInfo(), internal error, "+
                                                "exception raised:");
                                ProxyDebug.logException(ex);
                        }
                }

        }


        /*
         * Add the response to the user who wants to perform forwarding 22-1-2017
         */
        public synchronized void processUserForward(Request request, SipProvider sipProvider,
                        ServerTransaction serverTransaction ) {
                try{
                        MessageFactory messageFactory=proxy.getMessageFactory();

                        String key=getKey(request);

                        // Add the key if it is a new user:
                        if (ProxyDebug.debug){
                                ProxyDebug.println
                                ("Registrar, processUserForward(), key: \""+key+"\"");
                        }
                        if (key==null){
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println
                                        ("Registrar, processUserForward(), key is null"+
                                                        " 400 INVALID REQUEST replied");
                                }
                                Response response=messageFactory.createResponse
                                                (Response.BAD_REQUEST,request);
                                if (serverTransaction!=null)
                                        serverTransaction.sendResponse(response);
                                else sipProvider.sendResponse(response);
                                return ;
                        }


                        if ( registrationsTable.hasRegistration(key) ) {
                                ProxyDebug.println("Content request sou einai: "+request.getContent());
                                ProxyDebug.println(" to RequestURI sou einai: "+request.getRequestURI());
                                ProxyDebug.println("To key tha einai: "+key.toString());

                                String ForwardTo = null;
                                boolean hasCycles = false;
                                String Sender = key.toString();
                                if(request.getMethod().equals("FORWARD")){
                                        ForwardTo = request.getRequestURI().toString();
                                        hasCycles = foundCycleInRegistrationsGraph(Sender, ForwardTo);
                                }


                                //FIXED insert users that do not exist in list
                                boolean forwardeeIsValid = false;
                                if (ForwardTo!=null && registrationsTable.hasRegistration(ForwardTo)){
                                        forwardeeIsValid = true;
                                }
                                else if(ForwardTo==null){
                                        forwardeeIsValid = true;
                                }
                                else{
                                        forwardeeIsValid = false;
                                }

                                // Here we should validate if the update in the registration table by checking the graph


                                if(!hasCycles && forwardeeIsValid){
                                        boolean updateresult = registrationsTable.updateForwardRegistration(Sender, ForwardTo);

                                        if (updateresult){
                                                //if we are here everything went as planned
                                                Response response=
                                                                messageFactory.createResponse(Response.OK,request);


                                                if (serverTransaction!=null)
                                                        serverTransaction.sendResponse(response);
                                                else sipProvider.sendResponse(response);

                                                if (ProxyDebug.debug)  {
                                                        ProxyDebug.println
                                                        ("Registrar, processUserForward(), response sent:");
                                                        ProxyDebug.print(response.toString());
                                                }

                                                // if registrations table has changed we should update the registrations.xml file
                                                this.writeXMLRegistrations();

                                                return;

                                        }

                                }
                        }


                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("No valid User To forward");
                        }
                        Response response=messageFactory.createResponse
                                        (Response.BAD_REQUEST,request);
                        if (serverTransaction!=null)
                                serverTransaction.sendResponse(response);
                        else sipProvider.sendResponse(response);
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserForward(), response sent:");
                                ProxyDebug.print(response.toString());
                        }
                        return ;


                } catch (SipException ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println("Registrar.processUserForward exception raised:");
                                ProxyDebug.logException(ex);
                        }
                } catch(Exception ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserForward(), internal error, "+
                                                "exception raised:");
                                ProxyDebug.logException(ex);
                        }
                }
        }


        /*
         * Add the response to the user who wants to perform Block 25-1-2017
         */
        public synchronized void processUserBlocking(Request request, SipProvider sipProvider,
                        ServerTransaction serverTransaction ) {
                try{
                        MessageFactory messageFactory=proxy.getMessageFactory();

                        String key=getKey(request);

                        // Add the key if it is a new user:
                        if (ProxyDebug.debug){
                                ProxyDebug.println
                                ("Registrar, processUserBlocking(), key: \""+key+"\"");
                        }
                        if (key==null){
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println
                                        ("Registrar, processUserBlocking(), key is null"+
                                                        " 400 INVALID REQUEST replied");
                                }
                                Response response=messageFactory.createResponse
                                                (Response.BAD_REQUEST,request);
                                if (serverTransaction!=null)
                                        serverTransaction.sendResponse(response);
                                else sipProvider.sendResponse(response);
                                return ;
                        }


                        if ( registrationsTable.hasRegistration(key) ) {
                                ProxyDebug.println("Content of request is: "+request.getContent());
                                ProxyDebug.println("URI of request is: "+request.getRequestURI());
                                ProxyDebug.println("Key is: "+key.toString());

                                boolean updateresult = false;

                                String BlockID = null;
                                String Sender = key.toString();
                                if(request.getMethod().equals("BLOCK")){
                                        BlockID = request.getRequestURI().toString();

                                        //FIXED insert users that do not exist in list
                                        if (registrationsTable.hasRegistration(BlockID)){
                                                updateresult = registrationsTable.insertToBlockedUsersListRegistration(Sender, BlockID);
                                        }
                                        else {
                                                updateresult = false;
                                        }
                                }

                                if(request.getMethod().equals("UNBLOCK")){
                                        BlockID = request.getRequestURI().toString();
                                        updateresult = registrationsTable.deleteFromBlockedUsersListRegistration
                                                        (Sender, BlockID);
                                }


                                if (updateresult){
                                        //if we are here everything went as planned
                                        Response response=
                                                        messageFactory.createResponse(Response.OK,request);


                                        if (serverTransaction!=null)
                                                serverTransaction.sendResponse(response);
                                        else sipProvider.sendResponse(response);

                                        if (ProxyDebug.debug)  {
                                                ProxyDebug.println
                                                ("Registrar, processUserBlocking(), response sent:");
                                                ProxyDebug.print(response.toString());
                                        }
                                        // if registrations table has changed we should update the registrations.xml file
                                        this.writeXMLRegistrations();

                                        return;

                                }


                        }


                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("No valid User To Block");
                        }
                        Response response=messageFactory.createResponse
                                        (Response.BAD_REQUEST,request);
                        if (serverTransaction!=null)
                                serverTransaction.sendResponse(response);
                        else sipProvider.sendResponse(response);
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserBlocking(), response sent:");
                                ProxyDebug.print(response.toString());
                        }
                        return ;


                } catch (SipException ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println("Registrar.processUserBlocking exception raised:");
                                ProxyDebug.logException(ex);
                        }
                } catch(Exception ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserBlocking(), internal error, "+
                                                "exception raised:");
                                ProxyDebug.logException(ex);
                        }
                }
        }

        public boolean foundCycleInRegistrationsGraph(String newV1, String newV2){
                // check if the functionality does not allow autoregression
                ProxyDebug.println("[Registrations Graph here] Prior To initilization!");

                //intialize the graph object
                DirectedGraph<String, DefaultEdge> g = new DefaultDirectedGraph<String, DefaultEdge>(DefaultEdge.class);

                ProxyDebug.println("[Registrations Graph here] Initialized Graph!");

                Iterator iterator=registrationsTable.getRegistrations().keySet().iterator();

                ProxyDebug.println("[Registrations Graph here] Initialized Iterator!");

                while (iterator!=null && iterator.hasNext()) {
                        //ProxyDebug.println("[Registrations Graph Iteration] In!");

                        Registration registration=(Registration)registrationsTable.getRegistrations().get(iterator.next());
                        //ProxyDebug.println("[Registrations Graph Iteration] Got Registration! "+registration.toString());

                        String vertexToAdd = registration.getKey();
                        //ProxyDebug.println("[Registrations Graph Iteration] Vertex To add! "+vertexToAdd);

                        g.addVertex(vertexToAdd);
                        //ProxyDebug.println("[Registrations Graph Iteration] Vertex Added! ");

                }
                ProxyDebug.println("[Registrations Graph here] Vertexes Done!");


                boolean sameEdgeTwice = false;

                iterator=registrationsTable.getRegistrations().keySet().iterator();
                while (iterator!=null && iterator.hasNext()) {
                        Registration registration=(Registration)registrationsTable.getRegistrations().get(iterator.next());
                        String v1 = registration.getKey();
                        String v2 = registration.getForwardToUser();
                        ProxyDebug.println("[Edge Graph iteration] Checking Edge now: "+ v1 + "|" +v2);

                        if (v2 != null){
                                if (v1.equals(newV1) && v2.equals(newV2)){
                                        ProxyDebug.println("[Edge Graph iteration] edge already found: "+ v1 + "|" +v2);
                                        sameEdgeTwice = true;
                                }

                                ProxyDebug.println("[Edge Graph iteration] Inserting Edge now: "+ v1 + "|" +v2);
                                g.addEdge(v1,v2);
                        }
                }
                //ProxyDebug.println("[Registrations Graph here] foundCycleInRegistrationsGraph: "+ g);

                //if the edge does not already exists then add it in the graph and check if there is a circle
                if (!sameEdgeTwice){
                        ProxyDebug.println("[Edge Graph iteration] The edge has NOT been found, Add it now: "+ newV1 + "|" +newV2);
                        g.addEdge(newV1, newV2);
                }

                ProxyDebug.println("[Registrations Graph here] Built in visualization: "+ g);
            //JGraph jgraph = new JGraph( new JGraphModelAdapter( g ) );

                CycleDetector<String, DefaultEdge> cycleDetector = new CycleDetector<String, DefaultEdge>(g);
                ProxyDebug.println("[Registrations Graph here] Cycle Detector: ");

                boolean result = cycleDetector.detectCycles();
                ProxyDebug.println("[Registrations Graph here] Cycle Detector result: "+ result);


                return result;
        }

        public String findFinalForwardee(String key){
                Hashtable registrations=registrationsTable.getRegistrations();
        Registration currentRegistration=(Registration)registrations.get(key);

                String currentForwardee = currentRegistration.getForwardToUser();


                while (currentForwardee!=null){

                        currentRegistration = (Registration)registrations.get(currentForwardee);

                        if (currentRegistration == null){
                                return null;
                        }

                        currentForwardee = currentRegistration.getForwardToUser();
                }

                return currentRegistration.getKey();
        }

        public boolean foundInBlockedUsersList(String caller, String callee){
                boolean result = registrationsTable.inBlockedUsersListRegistration(caller,callee);
                return result;
        }

        /*
         * process the OPTION request
         */

        public void processUserBilling(Request request, SipProvider sipProvider,
                        ServerTransaction serverTransaction , HeaderFactory headerFactory){
                try{
                        MessageFactory messageFactory=proxy.getMessageFactory();

                        String content = new String( request.getRawContent());
                        String key = getKey(request);
                        String duration = content.split("Duration:")[1];

                        duration = duration.replace("\n","");

                        Integer durationTime = Integer.parseInt(duration);

                        Double durationTimeInSec = new Double(durationTime) / 1000;

                        ProxyDebug.println("Duration in Time in Seconds: "+durationTimeInSec+" for Key: "+key);
                        Double chargement = durationTimeInSec;



                        if (key==null ){
                                if (ProxyDebug.debug) {
                                        ProxyDebug.println
                                        ("Registrar, processUserBilling(), key is null"+
                                                        " 400 INVALID REQUEST replied");
                                }
                                Response response=messageFactory.createResponse
                                                (Response.BAD_REQUEST,request);
                                if (serverTransaction!=null)
                                        serverTransaction.sendResponse(response);
                                else sipProvider.sendResponse(response);
                                return ;
                        }

                        if ( registrationsTable.hasRegistration(key) ) {

                                Hashtable registrations=registrationsTable.getRegistrations();
                        Registration registration=(Registration)registrations.get(key);

                                ProxyDebug.println("Content of request is: "+request.getContent());
                                ProxyDebug.println("URI of request is: "+request.getRequestURI());
                                ProxyDebug.println("Key is: "+key.toString());

                                boolean updateresult = true;
```

```
                                if (updateresult){
                                        //if we are here everything went as planned
                                        Response response=messageFactory.createResponse(Response.OK,request);
                                        ContentTypeHeader hbill = headerFactory.createContentTypeHeader("application", "billing");

                                        if (registration.getuserCategory().equals("Normal")){
                                                chargement = durationTimeInSec * Chargement.normalChargement;
                                        }
                                        else if (registration.getuserCategory().equals("Premium")){
                                                chargement = durationTimeInSec * Chargement.premiumChargement;
                                        }

                                        String chargewritable = String.format( "%.2f",chargement);

                                        response.setContent("Chargement: "+chargement.toString(), hbill);

                                        //file for appending all the charges over the users
                                        String fileForCharges = "src/gov/nist/sip/proxy/configuration/UserCharges.txt";
                                        String textToAppend = "User: "+key+", Category: "+registration.getuserCategory()+
                                                        ", Chargement: " + chargewritable;
                                        this.appendFile(fileForCharges, textToAppend);

                                        if (serverTransaction!=null)
                                                serverTransaction.sendResponse(response);
                                        else sipProvider.sendResponse(response);

                                        if (ProxyDebug.debug)  {
                                                ProxyDebug.println
                                                ("Registrar, processUserBilling(), response sent:");
                                                ProxyDebug.print(response.toString());
                                        }

                                        return;

                                }

                        }


                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("No valid User To Charge the Bill");
                        }
                        Response response=messageFactory.createResponse
                                        (Response.BAD_REQUEST,request);
                        if (serverTransaction!=null)
                                serverTransaction.sendResponse(response);
                        else sipProvider.sendResponse(response);
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserBilling(), response sent:");
                                ProxyDebug.print(response.toString());
                        }
                        return ;


                } catch (SipException ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println("Registrar.processUserBilling exception raised:");
                                ProxyDebug.logException(ex);
                        }
                } catch(Exception ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processUserBilling(), internal error, "+
                                                "exception raised:");
                                ProxyDebug.logException(ex);
                        }
                }
        }



        //Function for debug purposes for a simple proxy response sender
        /*
        public synchronized void processUserForward2(Request request, SipProvider sipProvider,
                        ServerTransaction serverTransaction ) {
        ProxyDebug.println("Kalispera paw gia na steilw");

                try{
                        MessageFactory messageFactory=proxy.getMessageFactory();
                        String key=getKey(request);

                                Vector contactHeaders=getContactHeaders(key);
                                Response response=
                                messageFactory.createResponse(Response.OK,request);
                                try{
                                        if ( hasExpiresZero(request) ) {
                                                response.addHeader(request.getHeader(ExpiresHeader.NAME));
                                        }
                                }
                                catch(Exception e){
                                        e.printStackTrace();
                                }
                                if ( contactHeaders!=null ) {
                                        for (int i = 0; i < contactHeaders.size(); i++) {
                                                ContactHeader contact = (ContactHeader)
                                                                contactHeaders.elementAt(i);
                                                response.addHeader(contact);
                                        }
                                }

                        ProxyDebug.println("Ligo prin steilw");

                                if (serverTransaction!=null)
                                        serverTransaction.sendResponse(response);
                                else sipProvider.sendResponse(response);

                                if (ProxyDebug.debug)  {
                                        ProxyDebug.println
                                        ("Response to FOrward klasopatata, response sent:");
                                        ProxyDebug.print(response.toString());
                                }
                                return;
                        }
                catch (SipException ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println("Registrar exception raised:");
                                ProxyDebug.logException(ex);
                        }
                } catch(Exception ex) {
                        if (ProxyDebug.debug) {
                                ProxyDebug.println
                                ("Registrar, processRegister(), internal error, "+
                                                "exception raised:");
                                ProxyDebug.logException(ex);
                        }
                }
        }
        */

}
```

modified: sip-proxy/src/gov/nist/sip/proxy/registrar/Registration.java

```
@ Registration.java:22 @ import gov.nist.sip.proxy.*;
 * @version 1.0
 */
public class Registration {

        // extra fields for forward and block
        protected String ForwardToUser;
        protected Vector BlockedUsersList;

        //extra field for category of user
        public String userCategory;
        // extra field for password of the user
        protected String password;

    protected FromHeader fromHeader;
    protected ToHeader toHeader;
@ Registration.java:46 @ public class Registration {
    public Registration() {
        toExport=true;
        contactsList=new Vector();
        buddyList = new Vector();
        buddyList = new Vector();

        //initialization
        BlockedUsersList = new Vector();
        setForwardToUser (null);
        setuserCategory ("Normal");
        setPassword("");
    }

    protected ExportedBinding exportBinding() {
@ Registration.java:75 @ public class Registration {

    }

    public Vector getBlockedUsersList() {
        return BlockedUsersList;
    }

    public void setBlockedUsersList( Vector UserList) {
        this.BlockedUsersList = UserList;
    }

    public String getForwardToUser() {
        return ForwardToUser;
    }

    public void setForwardToUser( String User) {
        this.ForwardToUser = User;
    }

    public String getuserCategory () {
        return userCategory;
    }

    public void setuserCategory( String categ) {
        this.userCategory = categ;
    }

    public Vector getContactsList() {
        return contactsList;
    }
@ Registration.java:107 @ public class Registration {
        this.contactsList=contactsList;
    }

    public String getPassword () {
        return password;
    }

    public void setPassword( String passw) {
        this.password = passw;
    }

    public void addContactHeader(ContactHeader contactHeader) {
       contactsList.addElement(contactHeader);
    }
@ Registration.java:162 @ public class Registration {

    }

    public boolean insertToBlockedUsersListRegistration(String NewBlockedUser){
        boolean result = false;

        String newDesignatedBlockUser = NewBlockedUser;
        Vector BlockedList = this.getBlockedUsersList();

        //valid user to insert into registration
        if ( newDesignatedBlockUser!=null ){

                if (BlockedList != null){
                        // check if this block is inside the list
                Iterator itr = BlockedList.iterator();
                String CurrentBlockedUser = null;
                while (itr.hasNext()){
                        CurrentBlockedUser = itr.next().toString();

                        if (CurrentBlockedUser.equals(newDesignatedBlockUser)){
                                result = false;
                                return result;
                        }
                }
                }


                BlockedList.add(newDesignatedBlockUser);
                this.setBlockedUsersList(BlockedList);
                result = true;
        }

        return result;
    }

    public boolean deleteFromBlockedUsersListRegistration( String NewBlockedUser){

        boolean result = false;

        String newDesignatedBlockUser = NewBlockedUser;
        Vector BlockedList = this.getBlockedUsersList();

        //valid user to insert into registration
        if ( newDesignatedBlockUser!=null && BlockedList != null){

                result = BlockedList.remove(newDesignatedBlockUser);

                this.setBlockedUsersList(BlockedList);
        }

        return result;
    }

    public void updateContactHeader(ContactHeader contactParameter) {

        Address addressParam=contactParameter.getAddress();
@ Registration.java:285 @ public class Registration {
            retval.append("display_name=\""+displayName+"\"");
        }

        retval.append(" uri=\""+key+"\" ");
        if (this.getuserCategory()!=null) {
            retval.append(" category=\""+this.getuserCategory()+"\" ");
        }

        if (this.getPassword()!=null) {
            retval.append(" password=\""+this.getPassword()+"\" ");
        }

        retval.append("uri=\""+key+"\"> ");

        for( int i=0; i<contactsList.size();i++) {
            retval.append("     <CONTACT ");
@ Registration.java:318 @ public class Registration {

        // Append the buddy list to the contact.
        for( int i=0; i<buddyList.size();i++) {
             retval.append(" <BUDDY  uri= \"").append(buddyList.elementAt(i).toString()).append("/>\n");
             retval.append(" <BUDDY  uri= \"").append(buddyList.elementAt(i).toString()).append("/> ");
        }
```

```
      // Append the new fields as well 27-1-2017 update
        if (this.getForwardToUser()!=null) {
            retval.append("<FORWARD_TO uri=\""+this.getForwardToUser()+"\"/> ");
        }


        if (this.BlockedUsersList != null){
                Iterator itr = this.BlockedUsersList.iterator();
                while (itr.hasNext()){
                        String CurrentBlockedUser = itr.next().toString();
                        retval.append("<BLOCKED_USER uri=\""+CurrentBlockedUser+"\"/> ");
                }
        }

        retval.append("</REGISTRATION>\n");
        retval.append("\n</REGISTRATION>\n");
        return retval.toString();
    }
```

modified: sip-proxy/src/gov/nist/sip/proxy/registrar/RegistrationsTable.java

```
@ RegistrationsTable.java:169 @ throws RemoteException
        FromHeader fromHeader = (FromHeader)request.getHeader(FromHeader.NAME);
        registration.fromHeader = fromHeader;


        //set password
        String content = new String( request.getRawContent());
        String pass = content;
        try{
                pass = content.split("Password:")[1];
                pass = pass.replace("\n","");
                registration.setPassword(pass);
        }
        catch(Exception e){
                e.printStackTrace();
                 ProxyDebug.println("Add Registration Exception");
        }



                ProxyDebug.println("Registration Request: " + request.toString());
        registrations.put(key,registration);
        ProxyDebug.println
        ("RegistrationsTable, addRegistration(), registration "+
@ RegistrationsTable.java:225 @ throws RemoteException
        ("RegistrationsTable, addRegistration(), registration "+
        " added for the key: "+key);

        //initialize forward and block values
        //registration.setBlockedUsersList(null);
        //registration.setForwardToUser(null);



        printRegistrations();

        updateGUI(registration,false);
@ RegistrationsTable.java:263 @ throws RemoteException
            }
        }
    }


    //Update REgistration does not update all values but ONLY CONTACT LIST!!!!!!!!
    public void updateRegistration(String key,Request request) throws Exception {
        ProxyDebug.println("RegistrationsTable, updateRegistration(), registration updated"+
        " for the key: "+key);
@ RegistrationsTable.java:383 @ throws RemoteException
            String keyTable=(String)e.nextElement();
            Registration registration=(Registration)registrations.get(keyTable);
            ProxyDebug.println("registered user: \""+keyTable+"\"");
            registration.print();
            ProxyDebug.println("Forward To User: \""+registration.getForwardToUser()+"\"");


            //ProxyDebug.println("Blocked Users List: \""+registration.getBlockedUsersList().toString()+"\"");
            Vector BlockedList = registration.getBlockedUsersList();
            ProxyDebug.println("Blocked Users List: ");
                // check if this block is inside the list
            if (BlockedList != null){
                Iterator itr = BlockedList.iterator();
                while (itr.hasNext()){
                        String CurrentBlockedUser = itr.next().toString();
                    ProxyDebug.println("        Blocked User: "+CurrentBlockedUser);
                }
            }

            ProxyDebug.println();
        }
        ProxyDebug.println("************************************************");
@ RegistrationsTable.java:427 @ throws RemoteException
        }
    }


    // Implement some functions in order to update the registration tables for Forward and Block Requests
    public boolean updateForwardRegistration(String key, String NewUserToForward){
        ProxyDebug.println("RegistrationsTable, updateForwardRegistration(), registration updated"+
                " for the key: "+key);

        boolean result = true;

        String newDesignatedUserToForward = NewUserToForward;
        Registration registration=(Registration)registrations.get(key);

        // TODO check if this Forward should be done (according to graph)
        registration.setForwardToUser(newDesignatedUserToForward);

        printRegistrations();
        //see if GUI is mandatory to be updated
        return result;
    }

    public boolean insertToBlockedUsersListRegistration(String key, String NewBlockedUser){

        String newDesignatedBlockUser = NewBlockedUser;
        Registration registration=(Registration)registrations.get(key);
        return registration.insertToBlockedUsersListRegistration(NewBlockedUser);
    }


    public boolean deleteFromBlockedUsersListRegistration(String key, String NewBlockedUser){
        ProxyDebug.println("RegistrationsTable, deleteFromBlockedUsersListRegistration(), "
                        + "registration updated"+" for the key: "+key);

        String newDesignatedBlockUser = NewBlockedUser;
        Registration registration=(Registration)registrations.get(key);
        return registration.deleteFromBlockedUsersListRegistration(NewBlockedUser);
    }

    public boolean inBlockedUsersListRegistration(String key, String blockedUser){
        Registration registration=(Registration)registrations.get(key);
        ProxyDebug.println("inBlockedUsersListRegistration registered user: "+key+"\"");

        //ProxyDebug.println("Blocked Users List: \""+registration.getBlockedUsersList().toString()+"\"");
        Vector BlockedList = registration.getBlockedUsersList();

        // check if this blocked is inside the list
        if (BlockedList != null){
                Iterator itr = BlockedList.iterator();
                while (itr.hasNext()){
                        String CurrentBlockedUser = itr.next().toString();
                ProxyDebug.println("    Blocked User: "+CurrentBlockedUser);
                        if (CurrentBlockedUser.equals(blockedUser)){
                    ProxyDebug.println("FOUND HIM!!  "+CurrentBlockedUser);
                                return true;
                        }
                }
        }
        return false;
    }


}
```

modified: sip-proxy/src/gov/nist/sip/proxy/registrar/XMLRegistrationsParser.java

```
@ XMLRegistrationsParser.java:132 @ public class XMLRegistrationsParser extends DefaultHandler {
                 "the uri attribute has not been specified for the "+
                  "registration tag.");
            }
            String category=attrs.getValue("category");
            if (category!=null && !category.trim().equals("") ) {
                   registration.setuserCategory(category);
            }
            else {
                  ProxyDebug.println
                ("WARNING, XMLRegistrationsParser, startElement()"+
                " the userCategory attribute is not set for the registration tag!!!");
            }
            String password=attrs.getValue("password");
            if (password!=null && !password.trim().equals("") ) {
                   registration.setPassword(password);
            }
            else {
                  ProxyDebug.println
                ("WARNING, XMLRegistrationsParser, startElement()"+
                " the password attribute is not set for the registration tag!!!");
            }
        }

        if (element.compareToIgnoreCase("forward_to") ==0 ) {

            String forwardTo=attrs.getValue("uri");
            if (forwardTo!=null && !forwardTo.trim().equals("") ) {
                registration.setForwardToUser(forwardTo);
            }
            else {
                  ProxyDebug.println
                ("WARNING, XMLRegistrationsParser, startElement()"+
                " the forwardTo user is not set for the element forwardTo!!!");
            }
        }

        if (element.compareToIgnoreCase("blocked_user") ==0 ) {

                ProxyDebug.println("Papotsi: " + element);
            String blockUser=attrs.getValue("uri");
            if (blockUser!=null && !blockUser.trim().equals("") ) {
                ProxyDebug.println("Papotsi: " + blockUser);
                registration.insertToBlockedUsersListRegistration(blockUser);
                ProxyDebug.println("Papotsi: " + registration.getBlockedUsersList().toString());
            }
            else {
                  ProxyDebug.println
                ("WARNING, XMLRegistrationsParser, startElement()"+
                " the uri at the blocked_user element is wrong!!!");
            }
        }
```