



**IP-FP6-015964**

**AEOLUS**

**Algorithmic Principles for Building Efficient Overlay Computers**

**Deliverable D6.2.11**

**Trust-X prototype**

**An Overview of the Architecture of Trust-X Trust Management System**

---

Responsible Partner: University of Ioannina (EL)  
Report Preparation Date: February 2010

Contract Start Date: 01/09/05 Duration: 54 months  
Project Co-ordinator: University of Patras (EL)

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Architecture Overview</b>	<b>1</b>
2.1	Example of execution . . . . .	2
<b>3</b>	<b>Communication Layer</b>	<b>2</b>
<b>4</b>	<b>Database</b>	<b>5</b>
<b>5</b>	<b>Introductory Phase</b>	<b>8</b>
<b>6</b>	<b>Policy Evaluation Phase</b>	<b>8</b>
<b>7</b>	<b>Credential Exchange Phase</b>	<b>9</b>
<b>8</b>	<b>Negotiation Tree</b>	<b>9</b>
<b>9</b>	<b>Valid View</b>	<b>10</b>
<b>10</b>	<b>Groups</b>	<b>10</b>
10.1	Group Manager . . . . .	11
10.2	Group Agent . . . . .	11
<b>11</b>	<b>TrustXMembershipService</b>	<b>11</b>
11.1	Integration in the JXTAShell . . . . .	12

# 1 Introduction

Aim of this document is to provide an overview of the architectural design of the Trust- $\mathcal{X}$ , a comprehensive trust management system which supports complex trust negotiations with complex features like negotiations' crash recovery and voluntarily suspension and negotiations between groups.

Trust- $\mathcal{X}$  has been initially presented in [2]. During the recent years it has been extended in order to support trust negotiation recovery [6] and resource negotiations between peers' groups [5]. In order to support such features and to ease the integration of future extensions, the code of Trust- $\mathcal{X}$  prototype has been thoroughly refactored and in large parts completely rewritten. This work documents the architectural layout of the Trust- $\mathcal{X}$  prototype, as resulted from the refactoring and rewriting steps.

## 2 Architecture Overview

The Trust- $\mathcal{X}$  prototype has been implemented using Java 1.5. It supports the trust negotiation language presented in [6].

The source code has been organized in several packages according to the functionalities provided. Such packages are the following:

- **communicationLayer**, it provides a defined set of methods used by the upper layers to communicate with the other peers;
- **database**, it provides a standard interface for the database used in the different installations;
- **parser**, it provides a parser for the XML documents exchanged during the trust negotiation;
- **trustx**, it provides the classes required for the execution of a Trust- $\mathcal{X}$  negotiation. It also contains the packages of the different phases and the packages for some other features:
  - **gui**, it provides a simple graphical user interface to configure and execute a *client* peer of a Trust- $\mathcal{X}$  trust negotiation;
  - **introductoryPhase**, it provides the interface and the required classes to execute the introductory phase;
  - **policyEvaluation**, it contains interfaces and classes required to perform the policy evaluation phases;
  - **credentialExchange**, it provides the features required by the credential exchange phase;
  - **treemanager**, it contains the classes for the management of the negotiation tree;

- `groupmanager`, it contains the classes required to collaboratively verify the availability of the peers of the group and of the resources they stated to provided to the group.

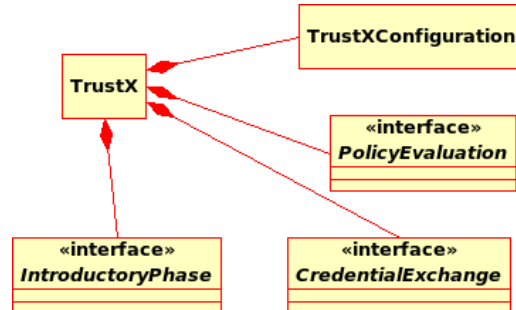


Figure 1: The high level class diagram of the Trust- $\mathcal{X}$  framework

The `parser` package contains the classes required for parsing, validating and creating the documents used in the trust negotiation and is therefore. It validates both  $\mathcal{X}$ -TNL and  $\mathcal{X}$ -RNL languages by means of a set of XML schemas.

To provide an easier way to create a trust negotiation, we implemented a Graphical User Interface (GUI) which provides a simple interface for the configuration and the the execution of a *client* peer for the Trust- $\mathcal{X}$ . Figure 2, 3 and 4 show respectively, the main window, how to configure the database connection and the communication layer which will be used.

## 2.1 Example of execution

An example on how to execute the Trust- $\mathcal{X}$  framework is provided by two targets within the ant config file (`build.xml`), contained in the source distribution. Such targets are `testServer` and `testClient`.

## 3 Communication Layer

The communication layer is contained in the `communicationLayer` package. Such package contains the interface `CommunicationLayer` which every real communication layer has to provide. Such interface is handled by the `CommunicationLayerManager`, which is in charge for returning the actual implementation of the configuration obtained as parameter of the static method `getCommunicationLayer()`.

The interface `CommunicationLayer` provides a core set of methods, necessary to allow the interactions of the parties involved in the trust negotiation. Such interface, shown in Figure 5, provides the following methods:

So far, there are three classes implementing the `CommunicationLayer` interface:

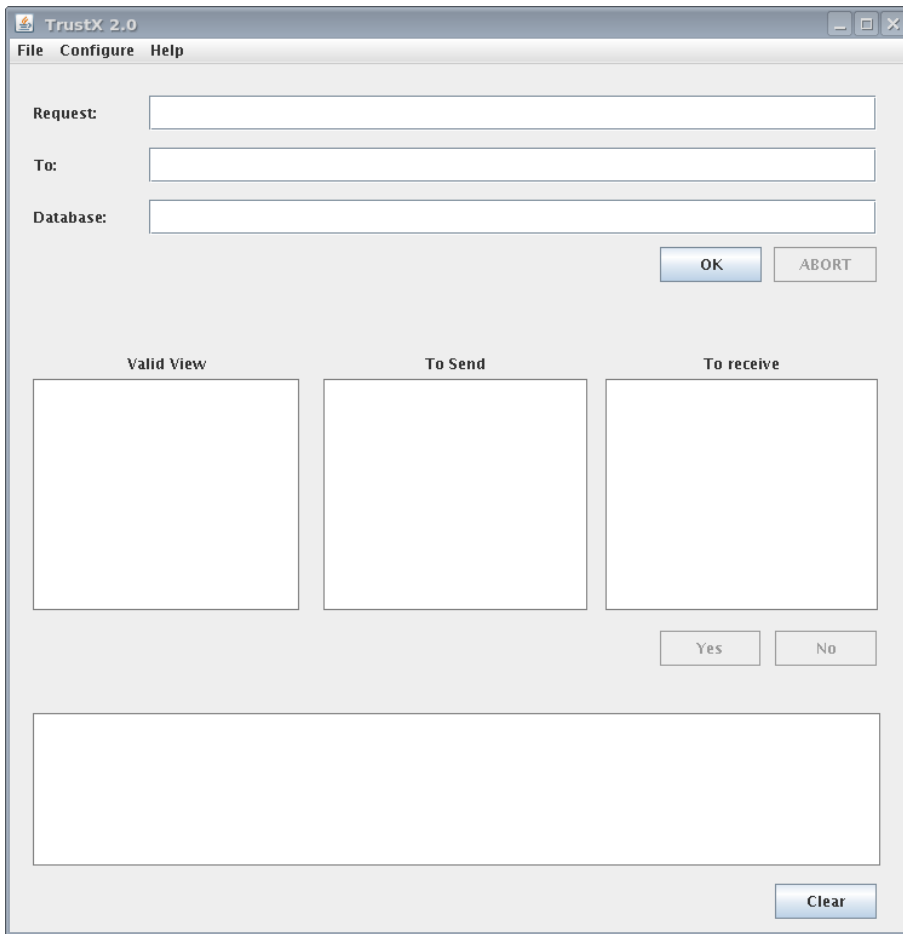


Figure 2: The main window

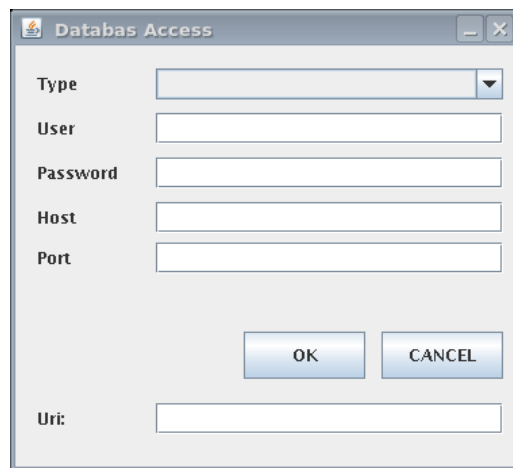


Figure 3: The dialog to configure the connection with the database

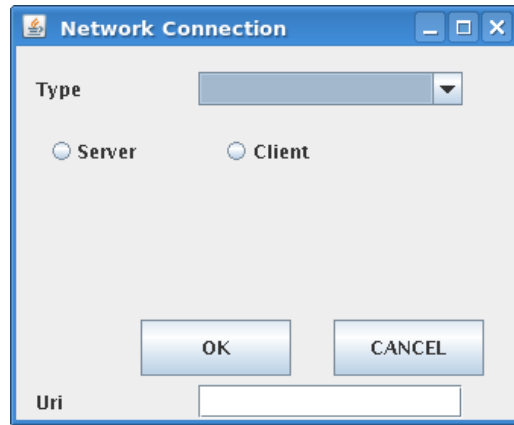


Figure 4: The dialog to configure the communication layer

```
public interface CommunicationLayer {  
    public Configuration getConfiguration ();  
    public void open() throws UnknownHostException,  
                IOException ;  
    public void close() throws IOException ;  
    public Message readMessage() throws IOException ;  
    public void writeMessage(Message msg)  
                throws IOException ;  
}
```

Figure 5: The interface `CommunicationLayer`

1. `CommunicationLayerOverJXTA`, a communication channel exploiting the JXTA network,
2. `CommunicationLayerOverIPv4`, a communication channel which uses TCP/IP socket, and
3. `CommunicationLayerOverIPv4SSL`, a communication layer which uses a socket connection protected with the SSL protocol.

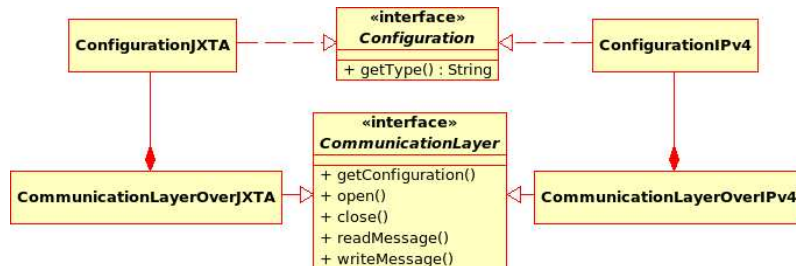


Figure 6: The class diagram of the package `communicationLayer`

## 4 Database

Besides the communication layer, an abstract class regarding the database used by parties has been written. This has been achieved by means of the interface `Database`. Such interface provides the methods required to retrieve the credentials and the policies used in a Trust- $\mathcal{X}$  negotiation.

In order to support a new DBMS, it is required to extend the `Configuration` class to handle the connection URI required by the DBMS.

By means of the method `getDatabaseInstance()`, the class `Configuration` will create an object of a class, which depends on the DBMS, implementing the `Database` interface.

The `Database` interface, shown in Figure 7, provides the method required to search and retrieve the resources present in the database and to retrieve the associated policies.

If the negotiation is performed not by two peers but by a peer and a group, the Group Manager, described in Section 10.1, needs to access a modified version of the `Database` interface. Such interface is called `DatabaseGroup`, shown in Figure 9, and is obtained by calling the method `getDatabaseGroupInstance()` of the `Configuration` class. The methods provided by the `DatabaseGroup` interface in addition to the ones provided by the `Database` interface have been introduced to perform searches of resources and policies among the ones belonging to the group.

The supported DBMS supported up to now are the following:

- MySQL[8] and

```

public interface Database{
    public void setConfiguration(Configuration conf)
        throws InvalidConfigurationException;
    public Configuration getConfiguration ();
    public void connect() throws NullPointerException ,
        IOException;
    public int searchResource(String str)
        throws ResourceNotFoundException ,
        IOException , CertificateException;
    public OurDocument getResource(int cid)
        throws ResourceNotFoundException ,
        IOException , CertificateException;
    public OurDocument searchAndGetResource(String str)
        throws ResourceNotFoundException ,
        IOException , CertificateException;
    public OurDocument readPolicy(int cid)
        throws IOException , CertificateException;
    public OurDocument searchAndReadPolicy(String resource)
        throws ResourceNotFoundException ,
        IOException , CertificateException;
}

```

Figure 7: The interface Database

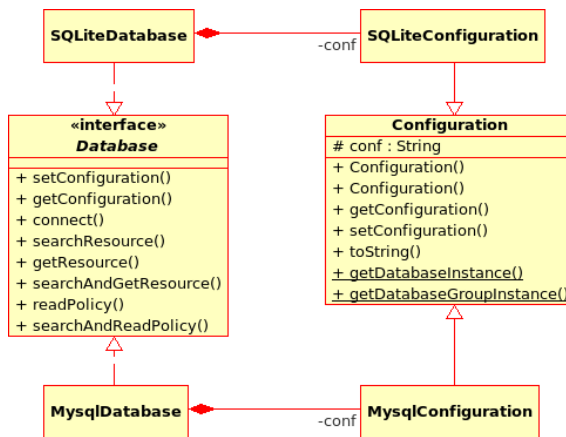


Figure 8: The class diagram for the database package

```

public interface DatabaseGroup {
    public void setConfiguration(Configuration conf)
        throws InvalidConfigurationException;
    public Configuration getConfiguration();
    public void connect() throws InvalidConfigurationException,
        IOException;
    public OurDocument searchAndReadPolicy(String resource)
        throws ResourceNotFoundException,
        IOException,
        CertificateException;
    public OurDocument readPolicy(int cid)
        throws IOException,
        CertificateException;
    public OurDocument searchAndReadGroupPolicy(String resource,
        int quantification)
        throws ResourceNotFoundException,
        IOException,
        CertificateException;
    public int addPeer(String uri) throws IOException;
    public int[] searchResources(String str, int quantified)
        throws ResourceNotFoundException,
        IOException, CertificateException;
    public int searchResource(String str) throws
        ResourceNotFoundException,
        IOException, CertificateException;
    public OurDocument getResource(int res)
        throws CertificateException, IOException;
    public int getOwnerResource(int res)
        throws IOException, CertificateException;
    public int addResource(String uri, OurDocument res,
        OurDocument policy, boolean necessary) throws IOException;
    public void addStrongRequest(int peerId, int resId)
        throws IOException;
    public void addStrongRequest(int peerId, int resId, int time)
        throws IOException;
    public boolean testStrongRequestValidity(int peerId, int resId)
        throws IOException;
    public DatabasePostconditions removePeer(String uri)
        throws IOException;
    public DatabasePostconditions removeResource(
        String uri, String resource) throws IOException;
    public int getPeersNumber() throws IOException;
}

```

Figure 9: The interface DatabaseGroup

- SQLite[1].

## 5 Introductory Phase

The introductory phase is the first phase of the trust negotiation provided by Trust- $\mathcal{X}$ . Such phase is in charge of performing the tasks required to synchronize the peers involved in the negotiation and, according to the implementation used in the current deployment, perform optional tasks such as find an agreement upon the resource to be negotiated, recover a previously suspended (both voluntarily and accidentally) negotiation and so on. The interface `IntroductoryPhase` provides a common set of methods to be used by the peer involved in the negotiation but each class implementing the `IntroductoryPhase` interface may provide more advanced methods.

The classes implementing such interface are the following:

- `DummyIntroductoryPhase`, a simple implementation created for debug purposes.
- `RebateIntroductoryPhase`, such class implements the rebate formulae during the introductory phase, as described in [3].
- `RecoveryIntroductoryPhase`, this class implements the resume/recovery phases described in [7].
- `StandardIntroductoryPhase`, this class implements all the above features. This implementation is the one which should be used in a real deployment of the Trust- $\mathcal{X}$  framework.

## 6 Policy Evaluation Phase

The policy evaluation phase occurs when each peer involved in the negotiation sends disclosure policies to the other peer upon the request of a resource. It terminates when the peers find a deliv subtree within the current negotiation tree (see Section 8 and [2]).

The methods required by such phase are provided by means of the interface `PolicyEvaluation`, as show in Figure 10.

```
public interface PolicyEvaluation {
    public void evaluateTree(Tree tree) throws IOException;
    public boolean searchAnotherValidView() throws IOException;
    public List<Tree> getLastValidViews();
    public boolean proposeValidViewC(boolean b) throws IOException;
    public boolean proposeValidViewS(boolean b) throws IOException;
}
```

Figure 10: The interface `PolicyEvaluation`

A negotiation can be performed between two single peers and between a peer and a group of peers. This fact required the creation of two distinct classes implementing the interface `PolicyEvaluation`.

A trust negotiation is normally a symmetric process, which means that both involved parties perform the same operations in order to execute the trust negotiation. The introduction of the possibility to perform a negotiation between a peer and a group of peers removes this symmetry. Therefore, the method required for a P2P negotiation has been implemented in the class `PolicyEvaluationImpl`. On the other side, for the SP2G negotiation three classes have been implemented. The first is the class `PolicyEvaluationGroup`, an abstract class providing the methods used by both the Group Manager and the external peer. Both the class executed by the GroupManager, called `PolicyEvaluationGroupManager`, and the class for the entering peer, called `PolicyEvaluationGroupPeer`, extend the abstract class `PolicyEvaluationGroup`.

## 7 Credential Exchange Phase

In the credential exchange phase the peers involved in the trust negotiation actually exchange the credentials and resources named in the valid view (see Section 9).

The method required for the execution of the credential exchange phase is provided by the interface `CredentialExchange`.

```
public interface CredentialExchange {
    public boolean exchange(Tree validView)
        throws IOException;
}
```

Figure 11: The interface `CredentialExchange`

Similarly to the policy evaluation phase, the operations really performed by a credential exchange phase slightly differ with respect of the type of negotiation performed. Again, it has been implemented a class for the negotiation P2P, called `CredentialExchangeImpl`, and three classes for the negotiation SP2G. The classes for the negotiation SP2G follow the same structure presented for the policy evaluation phase. The abstract class `CredentialExchangeGroup` implements the common methods and two classes extend such class in order to perform the operation specifically required by each party. Therefore, in the `trustx.credentialExchange` package there are the classes `CredentialExchangeGroupManager` and `CredentialExchangeGroupPeer`.

## 8 Negotiation Tree

The classes and interfaces contained in the `trustx.tree` package are the core of the trust negotiation performed by the Trust- $\mathcal{X}$  framework. They implement the negotiation tree,

which is the data structure containing credentials and (representation of) resources, and it is dynamically built by peers during the exchange of their disclosure policies.

A Trust- $\mathcal{X}$  negotiation tree is implemented by the class `Tree`, which extends the class `TreeManager`. The class `TreeManager` defines the interface used by the classes implementing the different negotiation phases and provides a set of methods common to each implementation of the negotiation tree. The class `Tree` provides an efficient implementation of the data structures required by the negotiation tree.

The negotiation tree evolves according to the policies exchanged by the parties. The terms which compose in the policies are represented within the negotiation tree by the class `Node`. Such class models the characteristic of a possible term as described in [4, 7, 5]. In case of extensions at the policy language, unless the modifications involve the structure of the policy, the class `Node` is the one which must be modified in order to implement the required features.

Moreover, the class `TreeManager` provides the methods required to save the negotiation tree to a stable storage and to later load the same negotiation tree from the stable storage.

## 9 Valid View

A valid view, as described in [2], is a subtree of the negotiation tree in which every nodes are marked as *deliverable*<sup>1</sup>. The class `TreeManager` is able to identify if within the negotiation tree there is at least a subtree with such property. With the method `getValidView()` a user is able to obtain all the subtrees of the negotiation tree which are definable as valid view. With such list, a user may choose one of the available valid view and, instantiating an object of the class `ValidViewManager` can operate on the subtree as it were a *credential sequence*<sup>2</sup>.

## 10 Groups

As anticipated, the Trust- $\mathcal{X}$  framework has been extended in order to be able to handle trust negotiation between a peer and a group of peers. To obtain such type of negotiation and the advanced features introduced in the language  $\mathcal{X}$ -RNL, such as weak and strong formulae and quantifiers, some way to handle the state of the group are needed. The package `trustx.groupmanager` provides the interfaces and the classes required to handle the state of the group and the consistency of the strong formulae.

---

<sup>1</sup>We recall that a node is marked as deliverable if its disclosure policy is satisfied, i.e. if and only if it has no disclosure policy associated or there exist a set of terms in its disclosure policy marked as deliverable such that the whole disclosure policy holds[2]

<sup>2</sup>A credential sequence is the serialization of the nodes contained in a valid view

## 10.1 Group Manager

The manager of the group is the peer that created the group. The class `GroupManager` provides the methods required by the group manager in order to have an updated view of the group and to handle the consistency of the formulae marked as strong. We recall that a strong formula is required to be valid along the whole time period a peer is member of a group. The object of the class `GroupManager` must be instantiated by the group manager when it creates the group protected with the Trust- $\mathcal{X}$  framework and must be kept for the whole duration of the existence of the group.

## 10.2 Group Agent

The group agent is a thread that will remain in each peer of the group to communicate with the group manager in order to collaboratively determinate the state of the group. More precisely, at the end of the trust negotiation required in order to join the group, if the negotiation ends successfully, an object of the class `ConnectionToGroup` will be instantiated. Through the method provided by such class the agent will be able to communicate with the group manger in order to send it the modification of the state of the group the peer detects. This include even the modification caused by the peer itself such as when a peer revoke the availability of a resource provided or when it decides to leave the group.

The class `ConnectionToGroup` provides another important feature. Such feature is the *heartbeat*. The heartbeat is a ping sent by the group manager to each peer within the group. This ping is required in order to verify the liveness of each peer because each peer may crash or became unavailable. If such unavailability is protracted for more then an user define period, the group manager will mark the peer as definitively unavailable and will operate to remove the peer and its resources from the group pool.

## 11 TrustXMembershipService

To experiment the proposed solution, the JXTA framework has been extended by means of the creation of a new `MembershipService` which exploits the Trust- $\mathcal{X}$  framework.

To perform the two side authentication required by Trust- $\mathcal{X}$ , we also created an authentication service, implemented by means of the class `TrustXAuthService`. Such authentication service provides the server side of the Trust- $\mathcal{X}$  negotiation and is therefore associated with the group it is mandated to protect. To achieve that, such class implements the interface `Service`, which is required to create core JXTA services, and the interface `Runnable` in order to be able to execute the authentication service as a separate thread. Moreover, it provides the services required by the `GroupManager` described in Section 10.1.

The class `TrustXMembershipService` implements the interface `MembershipService`. Such interface defines the methods required by a `MembershipService` compliant to the JXTA architecture. Therefore, the implementing class provides the `apply` method. Such

```

public interface MembershipService {
    public Authenticator apply(AuthenticationCredential application) throws
        PeerGroupException, ProtocolNotSupportedException;
    public Credential join(Authenticator authenticated) throws PeerGroupException;
    public resign() throws PeerGroupException;
}

```

Figure 12: The interface `MembershipService`

method, upon the examination that the `AuthenticationCredential` received as parameter requires the execution of a join process which exploits Trust- $\mathcal{X}$ , returns an object implementing the interface `Authenticator`. In the specific case, the object is an instance of the class `TrustXAuthenticator`, described in what follows. Another method required by the interface `MembershipService` is the method `join`, which if the joining process ends successfully, returns an object of the class `TrustXCredential`, which contains the `GroupAgent` described in Section 10.2. Finally, the method `resign` is used to leave the group.

The class `TrustXAuthenticator` implements the interface `Authenticator`. An instance of such class is obtained by the invocation of the method `apply` of the class `TrustXMembershipService`. The `TrustXAuthenticator` is the class which implements the client side of a Trust- $\mathcal{X}$  negotiation. In order to perform such task, upon its creation, the instance must be instructed with the correct information such as the location of the database and the communication layer to use for the negotiation. Moreover, it implements all the methods required by the implemented interface `Authenticator`, which follow the behavior defined by the JXTA architecture.

```

public class TrustXAuthenticator {
    public void setTrustXDatabase(String dbUrl) throws
        NullPointerException, InvalidConfigurationException;
    public void useJXTAConnection() throws PeerGroupException;
    public void useIPV4Connection(String conn);
    ...
}

```

Figure 13: The methods of the class `TrustXAuthenticator`

All the classes described above, can be found in the package `net.jxta.impl.membership.trustx`

## 11.1 Integration in the JXTAShell

The `MembershipService` described in the previous part of the current section, has been made available to the user by means of commands of the JXTA shell. The following commands have been introduced:

- `trustx.newprgp`, such command operates similarly to the original command `newprgp`. It creates a new peergroup using the `TrustXMembershipService` described above as `MembershipService`.

- `trustx.join`, such command let a peer join a peergroup created with the `trustx.newgrp` command.
- `trustx.authsvc`, this command start and stop the TrustXAuthService for the current peergroup.
- `trustx.login`, which execute the authentication of the current peer in the joined peergroup. It has as side effect the execution of the GroupAgent within the current peer.

## References

- [1] SQLite database version 3. Available at <http://www.sqlite.org>.
- [2] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Trust- $\mathcal{X}$ : A Peer-to-Peer Framework for Trust Establishment. *IEEE Trans. Knowl. Data Eng.*, 16(7):827–842, 2004.
- [3] Stefano Braghin, Igor Nai Fovino, and Alberto Trombetta. Advanced trust negotiation in critical infrastructures. In *International Conference on Infrastructure Systems*, November 2008.
- [4] E. Bertino E. Ferrari, A.C. Squicciarini. X-tnl: An xml language for trust negotiations. *4th IEEE Workshop on Policies for Distributed Systems and Networks, Como, Italy*, June 2003.
- [5] Anna Cinzia Squicciarini, Federica Paci, Elisa Bertino, Alberto Trombetta, and Stefano Braghin. Group-Based Negotiations in P2P Systems. *Accepted by IEEE Transactions on Parallel and Distributed Systems*.
- [6] Anna Cinzia Squicciarini, Alberto Trombetta, and Elisa Bertino. Supporting robust and secure interactions in open domains through recovery of trust negotiations. In *ICDCS*, page 57, 2007.
- [7] Anna Cinzia Squicciarini, Alberto Trombetta, Elisa Bertino, and Stefano Braghin. Identity-based long running negotiations. In *Digital Identity Management*, pages 97–106, 2008.
- [8] Sun Microsystems. MySQL database version 5. Available at <http://www.mysql.com>.