# Peer To Peer Role Based Authentication within JXTA

95.495 Final Report

Ryan Laginski
266229

Supervisor: Tony White

April 11th, 2003

## Abstract:

To date, Peer-to-Peer networks (P2P) have difficulty determining an entity's identity. Authentication is a challenging aspect of P2P networks due to the lack of central authority. As such, authentication must be done distributely and use methods such as public/private keys, X.509 certificates or even simple flat XML files. The common belief that P2P is an anonymous method of communication and file distribution has led to many systems that consists of one type of peer. However, society has many different types of roles.

In this project, I intend to develop society based roles and integrate them with an authentication technique. I plan to implement this on top of an existing P2P framework, such as JXTA.

# Table of Contents

## Index Of Figures

## Index Of Tables

# 1 Introduction

JXTA (jxta.org) is a open source P2P framework initiated by Sun Microsystems (sun.com). As stated on their website: “JXTA technology is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner” (jxta.org). It was chosen for it's available documentation and books, ability to add in Services, and because a role-based authentication scheme does not exist.

First, a quick introduction to JXTA's terminology. A JXTA peer is “any entity capable of performing some useful work and communicating the results of that work to another eneity over a network...” (JXTA 16). A JXTA peergroup is “a set of peers formed to serve a common interest or goal dictated by the peers involved. Peergroups can provide services to their member peers that aren't accessible by other peers in the P2P network” (JXTA 18). Authentication in JXTA is defined by the process required in allowing a peer to join a peergroup after a provided set of credentials were verfied.

JXTA's current authentication scheme, as stated within the documentation, is not practical for production use. By default, the NullMembershipService is used, which allows anyone to join a peergroup. This scheme is similar to the nobody user in the UNIX system. The second implementation JXTA has is the PasswdMembershipService, which is more of a template for other developers. There exists no formal application process, as PasswdMembershipService will accept anyone who applies.

The goal of this project is to create  society based roles and integrate them with an authentication technique that would mimic real life scenarios. In real life terms, being connected to the JXTA network is similar to walking down a city street. People are all around you, but you do not know them or speak to every one. You can ask a person directions, in which case, the person would answer or ignore you. You can keep asking others until you get a response. With this example, you do not need to authenticate yourself, as your question, nor the response is secret information.

Suppose you are visiting a club were you are on the guest list. You would approach the doorman and he/she will ask you for your credentials. Provided you have the correct credentials, you may enter, otherwise, you are asked to leave. JXTA's peergroups, are analogous to a virtual rooms. This virtual room would contain people who share a common interest.

## 1.1 Limitations

To maintain the goal of this project, some limitation must be imposed. This project is not secure enough nor fully implemented. This design includes encryption which protects the information stored on disk and passed between peers. However, it's use is not fully implemented.

## 2 Proposed Design

In order to mimic real life roles in JXTA, a hiearchy of peertypes was created. There are four different roles, which are FounderPeer, SuperPeer, AuthenticationPeer and Peer. Each roles specifies a peer's capabilities within a peergroup. There is five capabilities, which are Override, Add, Delete, Modify, Authenticate. The following table explains the relationship between peertypes and their capabilities.

| ***Capable \ PeerType*** | ***FounderPeer*** | ***SuperPeer*** | ***Authentication Peer*** | ***Peer*** |
|---|---|---|---|---|
| Override | X | | | |
| Add | X | X | | |
| Delete | X | X | | |
| Modify | X | X | | |
| Authenticate | X | X | X | |

Table 1: Roles and Capabilities of the PeerTypes

A FounderPeer is the peer who created the peergroup. This peer determines the conditions of the peergroup, as described in the Uses Cases: Creating A Peer Group (p. 11). This peer is also able to override any decision, thus is able to add, delete, modify and authenticate any peer he/she wishes.

A SuperPeer is one step below the FounderPeer, and is able to add, delete, modify any account as well as authenticate a peer. There can be many SuperPeers, but generally, there will only be a few.

An AuthenticationPeer is only allowed to authenticate people based on a given list of credentials. Their role is similar to a doorman at the club entrance, however, since a virtual room has many entrances, there may be many AuthenticationPeers. The purpose of this role is to alleviate the load on Upper Members, which are SuperPeers and higher. This will also increase the odds that there will be a peer available to authenticate others.

A Peer's position is simply described as just a normal JXTA peer. They are not able to do anything administrative their position is defined by being members of the peer group.

## 2.1 Assumptions on the PeerTypes

There is only one FounderPeer. This assumption is to simplify the creation and maintainance of the peergroup.

The SuperPeers are well trusted by the FounderPeer, hence, there will be a small number of them.

The AuthenticationPeers are likely to be greater in quantity than the SuperPeers, however, they are not nearly as trusted as SuperPeers. This may seem to pose a risk, as anyone who is an AuthenicationPeer may allow anyone to join, regardless of the credentials provided. This issue is discussed in the Future Consideration (p. 25) section where SuperPeers should randomly check AuthenicationPeers.

# 3 Use Cases

The following is a collection of use cases that described the typical use of this project.

### 3.1 Scenario: Creating a peergroup

The first step in this project is to create a peergroup that requires authorization. This involves the FounderPeer to fill in the required information, such as encryption key size, how many SuperPeers are required to vote, are AuthenicationPeers allowed, etc. More details on this are described in the API section (p. 16). After the details are collect, a new AuthBasedPeerGroupAdvertisement is created and published.

### 3.2 Scenario: Join a Peergroup

There are two main scenarios in joining a peergroup. The first scenario is if the peer is an AuthenticationPeer or higher, it will send a request for peergroup advertisements. After the default period of time has past, the peer will check to see if the desired peergroup (that this upper member belongs to) exists. If it does, the AuthenicationPeer or higher will connect to it (as described below). If it does not exist, the AuthenticationPeer or higher will create and publish an advertisement for that peergroup. Once this occurs, there is a good chance that the same peergroup exists in several places within the network, but they are not visible to each other. The other scenario, the regular Peer must check their local advertisements for the group they wish to join, or a remote discovery is sent. If one is found, it can proceed with the connection, as described below. Otherwise, the peer must try again later.

The scenario begins when peer A attempts to establish a connection to the remote peer, B, that is apart from the peergroup. Peer B will check to see if it is capable of authenticating (ie, AuthenticationPeer or higher). Next, peer B will check the provided credentials. If the credentials are good, peer A has successfully joined the peergroup, otherwise, peer A is rejected. Also, if peer A is successful and is an Upper Member, it's presence is broadcast within the peergroup. If peer B is not capable of authenticating, peer B will send a list of known SuperPeers (if any) back so that peer A can try them.

## 3.3 Scenario: Account Request

The current JXTA implement of PasswdMembershipService provides an account to anyone who applies. This method is not practical in most situations because a group may want to keep their resources protected. A possible solution to this problem is have a peer request an account, then let the Upper Members of the peergroup vote on the request. This process starts by peer A connecting to a member within a peergroup and send it's desired credentials. The remote peer, B, will accept the AuthBasedPeerGroupCredentials if itself is an Upper Member. Details of the credentials is described in the API section (p. 16). If peer B is not an Upper Member, it will inform peer A to contact a SuperPeer and provide a list of known SuperPeers, from the presence updates.

Peer B will send a confirmation of application back to peer A.  If peer B is the FounderPeer, it may decide to grant or deny the account outright, as it has the override capability. The FounderPeer does have the option to have the account voted on. If peer B is a SuperPeer or is a FounderPeer that wishes to have the application voted on, the request is stored in peer B's local queue and a VoteOnAnAccountRequest is called.

## 3.4 Scenario: VoteOnAnAccountRequest

Once the application for an account is received, the SuperPeer peer locates X number of other superpeers for a vote, where X is the number specified by the FounderPeer upon peergroup creation. If the minimum number of SuperPeers are found, the SuperPeer will send the request (including the RequestID) to the X number of other SuperPeers. The SuperPeer will await responses to the vote, which is described below on VoteOnAnAccountResponse. A timeout may occur, and is handled by the Insufficient Number of SuperPeers Available scenario below (p. 14). Once all the votes have been received, the SuperPeer will determine if the account is granted or not. The peer is notified that it is accepted or rejected, however, if they are offline, the message is stored in the local queue. If it is an approval and the peer has received it's approval message, the peer is added to the local password file and that file is sent to the Replicator. If not all X SuperPeers are found, send the request to all available SuperPeers for them to queue the request. A vote will occur when sufficient SuperPeers are online.

## 3.5 Scenario: VoteOnAnAccountResponse

Once a SuperPeer receives an application, it will check to see if the application has the same RequestID as any in the local queue. If so, the SuperPeer will remove it from the local queue. Ideally, the SuperPeer will notify the user (human) about the application, however, for the scope of this project, a tester class will be hard coded with the answer. The SuperPeer then takes the answer, signs it with it's private key, and sends it to the originating SuperPeer.

## 3.6 Scenario: Insufficient Number of SuperPeers available

This scenario can be triggered by the a timeout while awaiting response, in the event a SuperPeer went offline without warning. The SuperPeer has already tried to discover other SuperPeers, but not enough are online. The SuperPeer will hold the request until enough SuperPeers advertise their presence. At that point, the SuperPeer will send request to all SuperPeers.

## 3.7 Scenario: Leaving the Peergroup

If peer is a SuperPeer, it's presence is updated as offline. This is required because an accurate list of online SuperPeers are used by all peers in determining the location of the SuperPeers. This list is sent to any peer that is trying to join or apply for an account and the remote peer does not have the capability to do so.

## 3.8 Scenario: Periodic Maintainance

If a SuperPeer has a message to send back to a peer in it's local queue, it will attempt to use the given PipeAdvertisement. If this fails, the SuperPeer will try to query for a new PipeAdvertisement. If this fails, the message is placed back in the queue. If the SuperPeer can find an advertisement, and the message from the queue is an approval, then the approval is sent to the peer and the peer is added to the local password file. That local password file is then sent to the Replicator.

After a request has gone through five attempts, whether or not it has been approved, disapproved or not yet voted on, it will be removed from the queue.

A SuperPeer must republish it's Presence every time it expires in JXTA's Discovery Service, so that an accurate list of SuperPeers is maintained.

Every so often, a SuperPeer should test AuthenicationPeers to make sure they are actually properly authenticating. This is described in Future Consideration (p. 25) section as Test Authentication Peers.

# 4 The API

This design is implemented over four services. The first service is AuthBasedPeerGroupService, which handles the apply and join methods, as required by Membership Service. The second service is the VoteOnAnAccountService, which is called by AuthBasedPeerGroupService's apply method and handles the voting. The third service is the SuperPeerPresenceService which handles the network status of the SuperPeers within an AuthBasedPeerGroup. The final service is the Replicator Service, which is responsible for password file distribution.

## 4.1 AuthBasedPeerGroupService

JXTA has interfaces for plugging in other authentication services. By default, any peergroup that is created will implement JXTA's NullMembershipService. Therefore, by replacing NullMembershipService with another implementation of authentication, one can take advantage of existing peergroup membership functionality. The membership service class in this implementation is called AuthBasedPeerGroupService.

To create an authentication peergroup, first, a new instance of AuthBasedPeerGroupService is created along with it's ModulesSpecID. A new AuthBasedPeerGroupAdvertisement is created with the details required for the AuthPeerGroupService. The details within the advertisement are shown in xml, which is encoded and decoded by AuthBasedPeerGroupAdvertisement.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<jxta:AuthBasedPeerGroupAdvertisement>
        <FounderID> . . . </FounderID>
        <KeySize> . . . </KeySize>
        <MinNumSuperpeersRequiredVote> . . . </MinNumSuperpeersRequiredVote>
        <MinNumSuperpeersRequiredApprove> . . . </ MinNumSuperpeersRequiredApprove>
        <IsAuthPeersAllowed> . . . </IsAuthPeersAllowed>
        <IsFullNameRequired>  . . . </IsFullNameRequired>
</jxta:AuthBasedPeerGroupAdvertisement>
```

Figure 1: A AuthBasedPeerGroupAdvertisement Message

The FounderID is a reference to the peer who created this particular peergroup. It can be used for purposes of communication or for determining other group interests. The KeySize is set by the FounderPeer at advertisement creation which specifies the key size for password file encryption, which is generally 1024. The MinNumSuperpeerRequiredVote represents the minimum number of SuperPeers required to vote on an application and is used during the VoteOnAnAccountRequest process. MinNumSuperpeerRequiredApprove represents how many SuperPeers are required for a vote to be successful. For example, four out of five SuperPeers are required, where five is MinNumSuperpeersRequiredVote. The IsAuthPeersAllowed field can enable or disable the AuthenicationPeer role, if the founder so chooses. This option, when enabled, adds additional security, as only the SuperPeers and the FounderPeer can authenticate peers. The downside is that since there is usually few SuperPeers and only one FounderPeer, the probability of them being offline is greater. The IsFullNameRequired field is optional and is purely for informational purposes.

Finally, the advertisement is published, and a new peergroup is cloned from the NetPeerGroup.

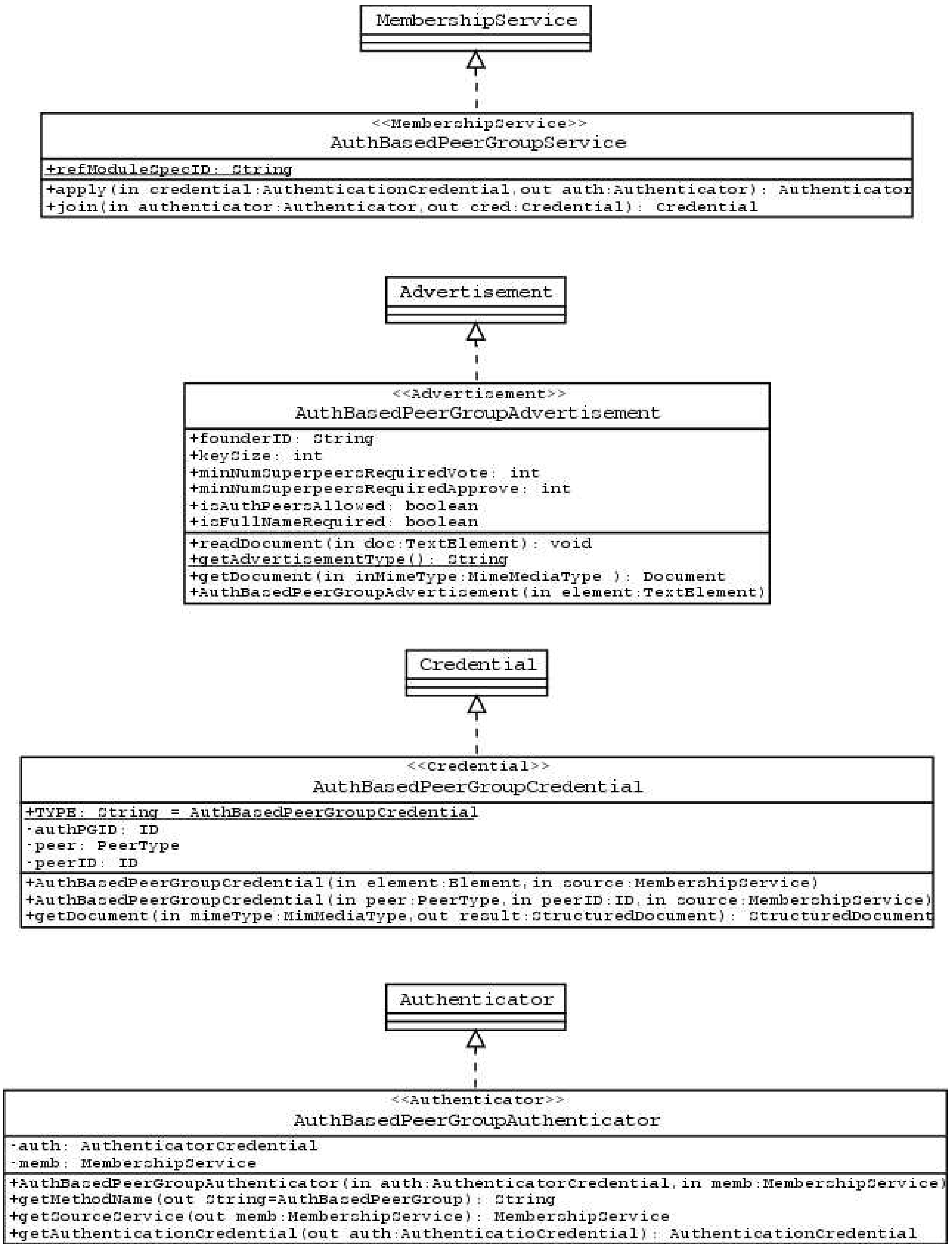
MembershipService
<<MembershipService>>
AuthBasedPeerGroupService
+refModuleSpecID: String
+apply(in credential:AuthenticationCredential,out auth:Authenticator): Authenticator
+join(in authenticator:Authenticator,out cred:Credential): Credential
Advertisement
<<Advertisement>>
AuthBasedPeerGroupAdvertisement
+founderID: String
+keySize: int
+minNumSuperpeersRequiredVote: int
+minNumSuperpeersRequiredApprove: int
+isAuthPeersAllowed: boolean
+isFullNameRequired: boolean
+readDocument(in doc:TextElement): void
+getAdvertisementType(): String
+getDocument(in inMimeType:MimeMediaType ): Document
+AuthBasedPeerGroupAdvertisement(in element:TextElement)
Credential
<<Credential>>
AuthBasedPeerGroupCredential
+TYPE: String = AuthBasedPeerGroupCredential
-authPGID: ID
-peer: PeerType
-peerID: ID
+AuthBasedPeerGroupCredential(in element:Element,in source:MembershipService)
+AuthBasedPeerGroupCredential(in peer:PeerType,in peerID:ID,in source:MembershipService)
+getDocument(in mimeType:MimMediaType,out result:StructuredDocument): StructuredDocument
Authenticator
<<Authenticator>>
AuthBasedPeerGroupAuthenticator
-auth: AuthenticatorCredential
-memb: MembershipService
+AuthBasedPeerGroupAuthenticator(in auth:AuthenticatorCredential,in memb:MembershipService)
+getMethodName(out String=AuthBasedPeerGroup): String
+getSourceService(out memb:MembershipService): MembershipService
+getAuthenticationCredential(out auth:AuthenticatioCredential): AuthenticationCredential

Figure 2: AuthBasedPeerGroupService UML Diagram

## 4.2 VoteOnAnAccountService

This service is based on JXTA's Query Handler to send the remote SuperPeer an application for an account. The application is outline in Figure 3 below.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<jxta:VoteOnAnAccountRequest>
        <RequestID>  . . . </RequestID>
        <FullName> . . . </FullName>
        <RequestingPosition> . . . </RequestingPosition>
        <PeerID> . . . </PeerID>
        <Nickname> . . . </Nickname>
        <Password> . . . </Password>
        <PublicKey> . . . </PublicKey>
        <Comment> . . . </Comment>
        <PipeAdvertisement> . . . </PipeAdvertisement>
</jxta:VoteOnAnAccountRequest>
```

Figure 3: A VoteOnAnAccountRequest Message

The FullName is optional, as specified within the peergroup's advertisement which is set by the FounderPeer. The RequestingPosition is the role the peer is applying for. PeerID is the the requesting peer's ID. The nickname is the name used by JXTA for it's peer name within a peergroup. The Password is an encrypted using the remote SuperPeer's peer public key, which is obtained from the presence service. The PublicKey is the peer's public key and comment is an optional string that is displayed to the SuperPeer. The PipeAdvertisement is used to send the peer back information, as below. Ideally, this information is sent over a secure pipe, so that information cannot be tampered with during transit.

Next, the remote peer will create a RequestID for this application. The RequestID is important, because it will be used as a reference by other Upper Members with the voting scheme. The voting procedure is described in the Use Cases (p 12)

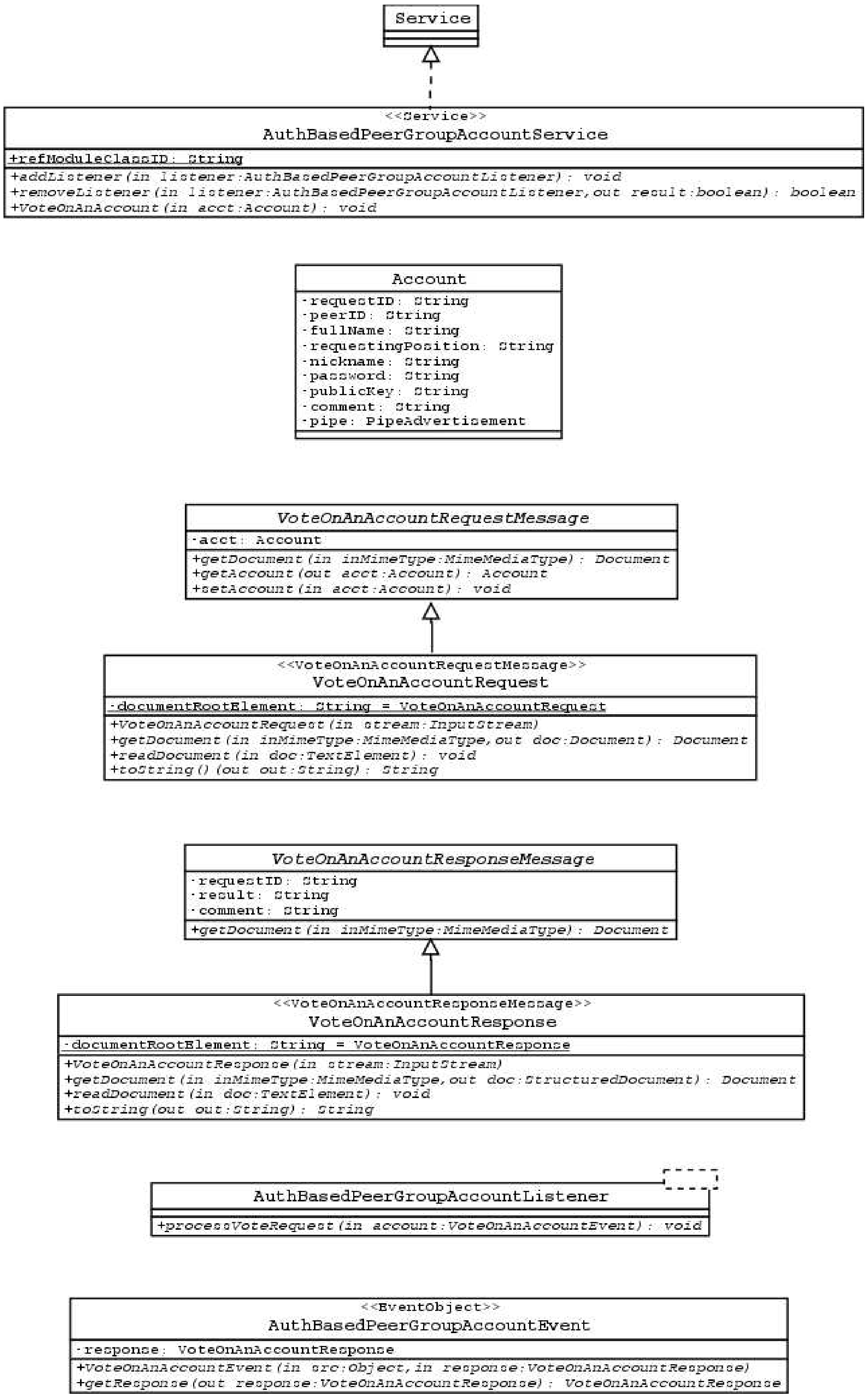
Service
<<Service>>
AuthBasedPeerGroupAccountService
+refModuleClassID: String
+addListener(in listener:AuthBasedPeerGroupAccountListener): void
+removeListener(in listener:AuthBasedPeerGroupAccountListener,out result:boolean): boolean
+VoteOnAnAccount(in acct:Account): void
Account
-requestID: String
-peerID: String
-fullName: String
-requestingPosition: String
-nickname: String
-password: String
-publicKey: String
-comment: String
-pipe: PipeAdvertisement
VoteOnAnAccountRequestMessage
-acct: Account
+getDocument(in inMimeType:MimeMediaType): Document
+getAccount(out acct:Account): Account
+setAccount(in acct:Account): void
<<VoteOnAnAccountRequestMessage>>
VoteOnAnAccountRequest
-documentRootElement: String = VoteOnAnAccountRequest
+VoteOnAnAccountRequest(in stream:InputStream)
+getDocument(in inMimeType:MimeMediaType,out doc:Document): Document
+readDocument(in doc:TextElement): void
+toString()(out out:String): String
VoteOnAnAccountResponseMessage
-requestID: String
-result: String
-comment: String
+getDocument(in inMimeType:MimeMediaType): Document
<<VoteOnAnAccountResponseMessage>>
VoteOnAnAccountResponse
-documentRootElement: String = VoteOnAnAccountResponse
+VoteOnAnAccountResponse(in stream:InputStream)
+getDocument(in inMimeType:MimeMediaType,out doc:StructuredDocument): Document
+readDocument(in doc:TextElement): void
+toString(out out:String): String
AuthBasedPeerGroupAccountListener
+processVoteRequest(in account:VoteOnAnAccountEvent): void
<<EventObject>>
AuthBasedPeerGroupAccountEvent
-response: VoteOnAnAccountResponse
+VoteOnAnAccountEvent(in src:Object,in response:VoteOnAnAccountResponse)
+getResponse(out response:VoteOnAnAccountResponse): VoteOnAnAccountResponse

Figure 4: AuthBasedPeerGroupAccountService UML Diagram

## 4.3 SuperPeerPresenceService

This presence service is required for determining how many and where the SuperPeers are located. This is used by the VoteOnAnAccountService when a SuperPeer must ask other SuperPeers to vote, and by AuthBasedPeerGroupService when a peer asks to join, but the peer is not capable of authenticating. This service uses JXTA's DiscoveryListener, which listens for SuperPeerPresenceEvent messages. Once an event has been received, the peer will decode the SuperPeer's ID, the status (offline or online), the SuperPeer's public key and PipeAdvertisement.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<jxta:SuperPeerPresenceAdvertisement>
        <PeerID> . . . </PeerID>
        <Status> . . . </Status>
        <PublicKey>  . . . </PublicKey>
        <PipeAdvertisement> . . . </PipeAdvertisement>
</jxta:SuperPeerPresenceAdvertisement>
```

Figure 5: A SuperPeerPresenceAdvertisement Message

To create an presence update, a peer will publish a new SuperPeerPresenceAdvertisement. Optionally, a peer can send a query to the discovery service to see if there is other superpeers that are online. This should only be done when a peer does not have enough SuperPeers to vote or when a peer needs to send another peer a list of SuperPeers, and has none. This will help reduce bandwidth, as this scenario will only occur if a peer recently connected and no SuperPeers have changed their status.

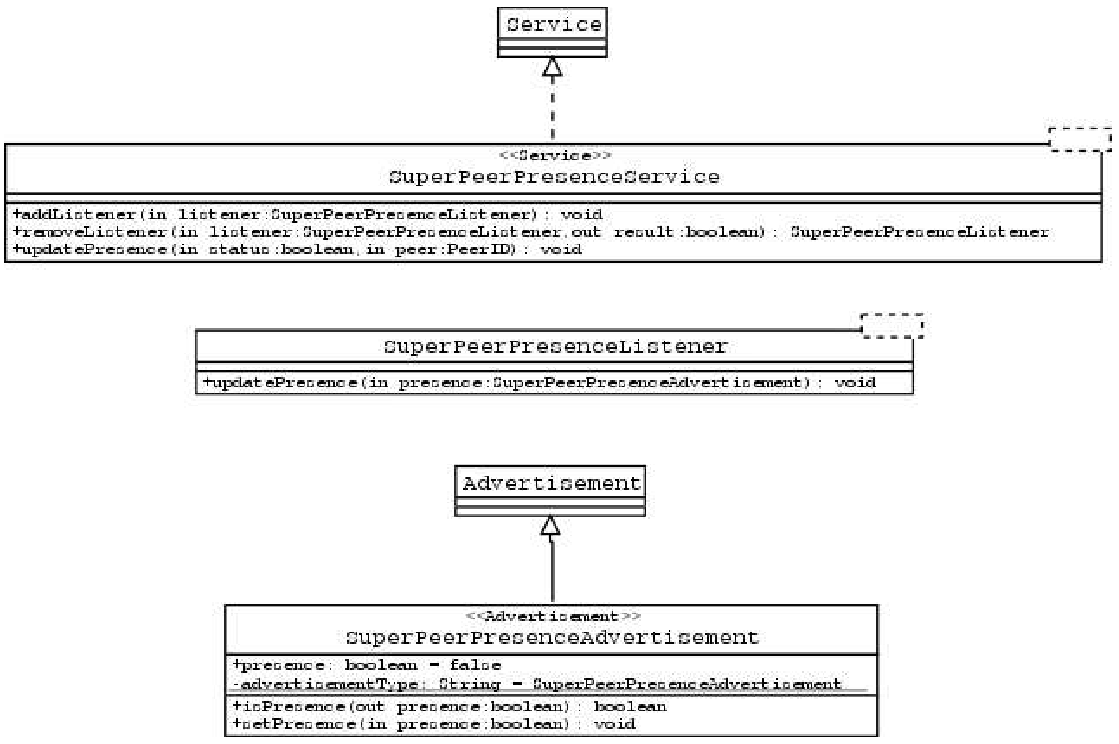
Service
<<Service>>
SuperPeerPresenceService
+addListener(in listener:SuperPeerPresenceListener) : void
+removeListener(in listener:SuperPeerPresenceListener,out result:boolean) : SuperPeerPresenceListener
+updatePresence(in status:boolean, in peer:PeerID) : void
SuperPeerPresenceListener
+updatePresence(in presence:SuperPeerPresenceAdvertisement) : void
Advertisement
<<Advertisement>>
SuperPeerPresenceAdvertisement
+presence: boolean = false
-advertisementType: String = SuperPeerPresenceAdvertisement
+isPresence(out presence:boolean) : boolean
+setPresence(in presence:boolean) : void

Figure 6: SuperPeerPresenceService UML Diagram

## 4.4 ReplicatorService

The ReplicatorService is used to check the password file with Upper Members. This service was not completely implementated, and is considered out of scope for this project. The Replicator contains three methods. The loadPasswdFile simply loads the password file from disk and returns a vector. This is used by checkVersionRequest and checkVersionResponse. CheckVersionRequest will check the presence service for available SuperPeers, as a check can only be done against a SuperPeer, and sends them a MD5 checksum and a time stamp  of their local password file.

CheckVersionResponse will determine if a file transfer is necessary, based on the received MD5 and time stamp. If so, requestFile is called and a file transfer is negotiated. A problem with the Replicator is that a new account may take awhile to propagate through the network, in which case, a peer may try to re-apply for an account. This scenario is noted in the Future Consideration section (p. 25).

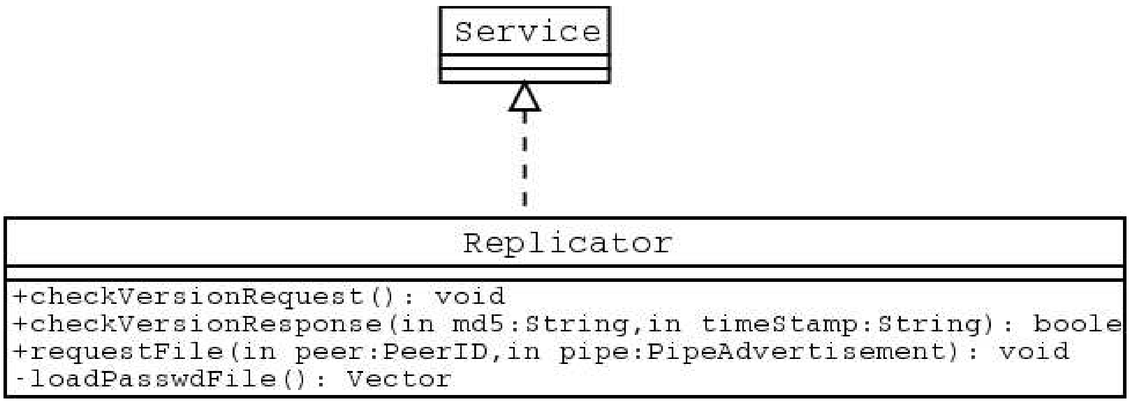
Service
Replicator
+checkVersionRequest(): void
+checkVersionResponse(in md5:String,in timeStamp:String): boole
+requestFile(in peer:PeerID,in pipe:PipeAdvertisement): void
-loadPasswdFile(): Vector

Figure 7: ReplicatorService UML Diagram

# 5 Design Decisions

- The peergroup information, as described in API AuthBasedPeerGroupService (p. 16), being stored in an advertisement can be seen as a security risk. Ideally, this information should not be public knowledge. By placing it in the advertisement, it made it easier to instantiate new AuthBasedPeerGroups.
- Due to the implementation of JXTA's MembershipService, a peer will send the credentials to the remote peer. However, in this implementation, the remote end may not be capable of authenticating or creating an account, so it will reject with a location of SuperPeers. The downside is that the remote peer receives the credentials by default, which is a security risk.
- The Presence Service does not keep track of AuthenicationPeers because they cannot grant accounts to the remote peer.
- Regular Peers cannot create the peergroup if it cannot find an advertisement for the desired AuthBasedPeerGroup, as there is no way for the peer to authenticate itself.
- For VoteOnAnAccount, the Account class created for decoupling, so that I can be replaced with another Account class that has more attributes.
- For the Replicator Service, only files can be retrieved from the Upper Members. This is to reduce the risk of a tampered password file from propagating, as Upper Members are more trustworthy.
- An approval does not immediately add the peer into the password file, rather, it ensures the peer receives the approval message first. After the peer has received the approval, the peer is added to the local password file, and the file is sent to the Replicator. This procedure is to prevent modifying the password file unneccessarily, as the SuperPeer may never 'see' the peer again.

- Since JXTA does not control who has phyical access to the computer, the password file is encrypted on disk to prevent people from tampering it's contains.
- The local queue is always loaded from disk and promptly saved to disk. This is done for backup purposes, in case the peer suddenly killed.

## 6 Problems Encountered

The Membership Service was very difficult to integrate with. It appears to have the assumption that all account applications will be granted. Extra information had to be passed with the credentials to deal with issues such as communicating back the the source peer. For example, if a peer attempt to apply to a Lower Member, it will have to send it a list of superpeers. However, the apply application returns an Authenticator, so null would be returned in the previous example.

It was also difficult to get JXTA to use the AuthBasedPeerGroupService instead of NullMembershipService. One of the main problems was that the ModuleSpecID must be generated based on PeerGroup's membershipClassID.

Another minor problem was with JXTA's caching scheme. The same bug that was previously fixed would reappear afterward. This was due to the old advertisement was still cache on a rendezvous server.

## 7 Future Consideration

The following is a list of features or ideas that were not implemented or completely thought out.

- Scenario: Test AuthenticationPeers

  Periodically, a upper member will 'test' the AuthenticationPeers for their credibility. The upper member will create a random credentials and attempt to join. If the peer allows the join, the AuthenticationPeer is removed from the password file. The is done by modifying the local password file and changing the position field, then replicating the password file. The problem is that the AuthenicationPeer is still capable of reading the password file because it has the key to decrypt it.

- To make this implementation fully functional, a complete Replicator Service and a GUI would be required.
- This implementation is not entirely secure. As noted in the Design Descions (p. 24), there exists several security issues that are unavoidable if the MembershipService is used. This design would have to abandon the Membership Service in order to become more secure.
- The encryption is not fully implemented in this design, nor in JXTA's PasswdMembershipService. JXTA's password protection is a simple character substitution cipher. In the future, this design could utilize a third party encryption scheme.
- The Replicator must find an efficient way of quickly distributing a new file. Otherwise, peers who request for an account and are approved may not be able to join for some time. The peer may try to reapply again, eventhough the credentials are already in a password file elsewhere.
- The Replicator must also not overwrite other peer's modifications. Since updates to the password file will be occurring at different areas of the network, it is likely that many different versions will exist.

## 8 Conclusion

This paper addresses the lack of society based roles within JXTA's peer-to-peer environment, by providing one of many possible solutions. In order to accomplish this, several different PeerTypes were designed to carry out specific capabilities within a group.

Unfortunately, JXTA's  Membership Service is not adequate for this proposed solution, and would be more secure and efficient by implementing it as a Service. However, this would require rewriting much of the JXTA PeerGroup code.

## 9 Glossary

Capability: A capability is a task that a peer can do. For example, a peer may be capable of authenticating a peer, but not adding one to the password file.

Local Queue: a Vector containing account applications and messages to hosts that are currently offline. The local queue is load from file whenever a peer requires it, and is saved to disk promptly.

Lower Members: refers to AuthenticationPeers and peers.

PeerType: A PeerType is reference to the capabilities that the peer posses. For example, a SuperPeer can add/delete/modify/authenticate a peer.

Presence: Used by all peers that belong to a peergroup, in determining the location of the SuperPeers.

Replicator: A black box that is in charge of sending a password file to peers that are AuthenicationPeer and above. Please see Figure 7 for more details.

Upper Members: refers to superpeers and the FounderPeer.

## 10 References

Wilson, Brendon J. JXTA, New Riders Publishing, Indianapolis, 2002.

Oaks, Traversat & Gong. JXTA in a Nutshell, O'Reilly & Associates, Inc., Sebastopol, 2002.

Oram, Andy ed. Peer-To-Peer: Harnessing the Power of Disruptive Technologies, O'Reilly & Associates, Inc., Sebastopol, 2001

Online Software Resources:

Project JXTA <http://jxta.org/>
Eclipse Project <http://www.eclipse.org/eclipse/index.html>
Sun JDK 1.4 <http://java.sun.com/j2se/1.4/index.html>