External RA Server Interface

06-07-31

1 Introduction/Scope

This document describes the design suggestions regarding the interface between EJBCA and the external RA Server.

For security reasons is it preferable to deny all inbound traffic to the CA and instead let the CA periodically fetch and process information from external trusted data sources. For an overview of the intended solution see illustration 1.

This document will emphasize on describing the API used between the developers of the RA server and the EJBCA Service.

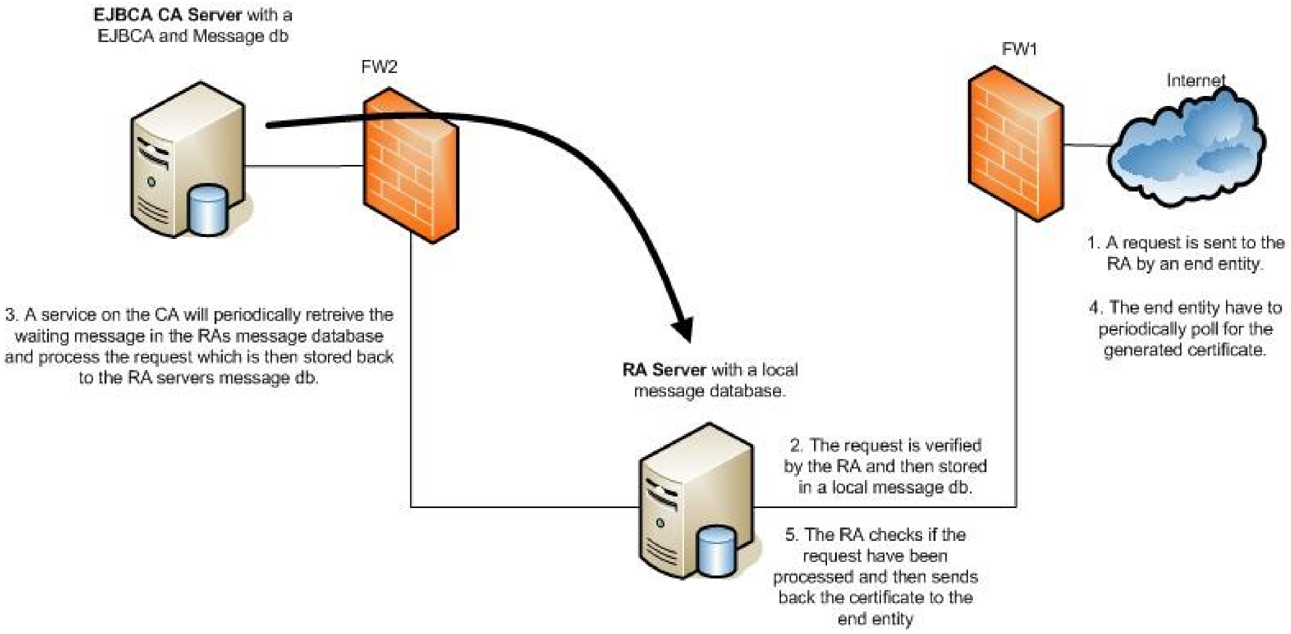

EJBCA CA Server with a EJBCA and Message db
FW2
FW1
Internet
1. A request is sent to the RA by an end entity.
4. The end entity have to periodically poll for the generated certificate.
3. A service on the CA will periodically retreive the waiting message in the RAs message database and process the request which is then stored back to the RA servers message db.
RA Server with a local message database.
2. The request is verified by the RA and then stored in a local message db.
5. The RA checks if the request have been processed and then sends back the certificate to the end entity

Illustration 1: Overview of solution

2 Document History

Version	*Date*	*Name*	*Comment*
1.0	2006-01-27	Philip Vendil	Initial version of this document.
2.0	2006-02-10	Philip Vendil	Added API and Test chapters
2.1	2006-02-14	Philip Vendil	Added authorization section
2.2	2006-02-24	Philip Vendil	Added revocation functionality
2.3	2006-07-02	Philip Vendil	Added edit user functionality
2.4	2006-07-31	Philip Vendil	Changed all se.primekey references to org.ejbca

Table of Contents

3 The Request Process

The processing of a end entity request is described in the sequence diagram in illustration 2. It defines three sub-processes: End Entity Request Process, ExtRA CA Service Process and End Entity Response Process.

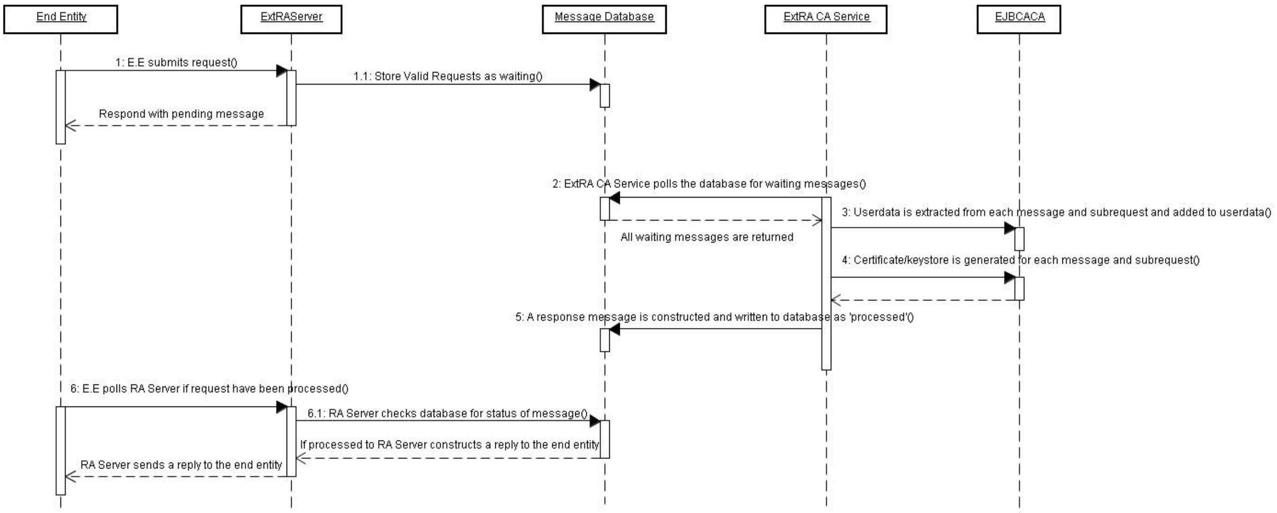

End Entity
ExtRAServer
Message Database
ExtRA CA Service
EJBCACA
1: E.E submits request()
1.1: Store Valid Requests as waiting()
Respond with pending message
2: ExtRA CA Service polls the database for waiting messages()
All waiting messages are returned
3: Userdata is extracted from each message and subrequest and added to userdata()
4: Certificate/keystore is generated for each message and subrequest()
5: A response message is constructed and written to database as 'processed'()
6: E.E polls RA Server if request have been processed()
6.1: RA Server checks database for status of message()
If processed to RA Server constructs a reply to the end entity
RA Server sends a reply to the end entity

Illustration 2: Sequence Diagram over Issuing process

3.1 End Entity Request Process

This process (point 1 in diagram) shows what happens when an end entity submits a request for certificates to the RA server. The RA server will review and verify the contents of the and create a 'Message' in a local database. This message contains information about the end entities requests like subjectDN, certificate type and expected response (Certificate, PKCS12, key recovered PKCS12) and so on. One message can contain several requests for each user, i.e both signature and authentication/encryption certificates can exist in one message to the CA server.

In this process should RA server respond with some form of 'request pending' message to the end entity.

3.2 ExtRA CA Service Process

This is a process run on the EJBCA server. (Point 2-5) It will periodically check for waiting messages on the RA server, fetch them and generate the appropriate responses for each user and store the message back to the RA server with status 'processed'.

3.3 End Entity Response Process

The last sub-process is when the end entity polls the RA server to see if the request have been generated. Here should the RA server check the database if the users message have status 'processed' and the generate an appropriate response back to the end entity. It the message is still in state 'waiting' or 'in process' should another 'request pending' message be returned.

4 Design of Message API

4.1 DB related Classes

The database related API will mainly consist of two classes, MessageHome which is used to manipulate messages with the database and the Message class representing an actual message.

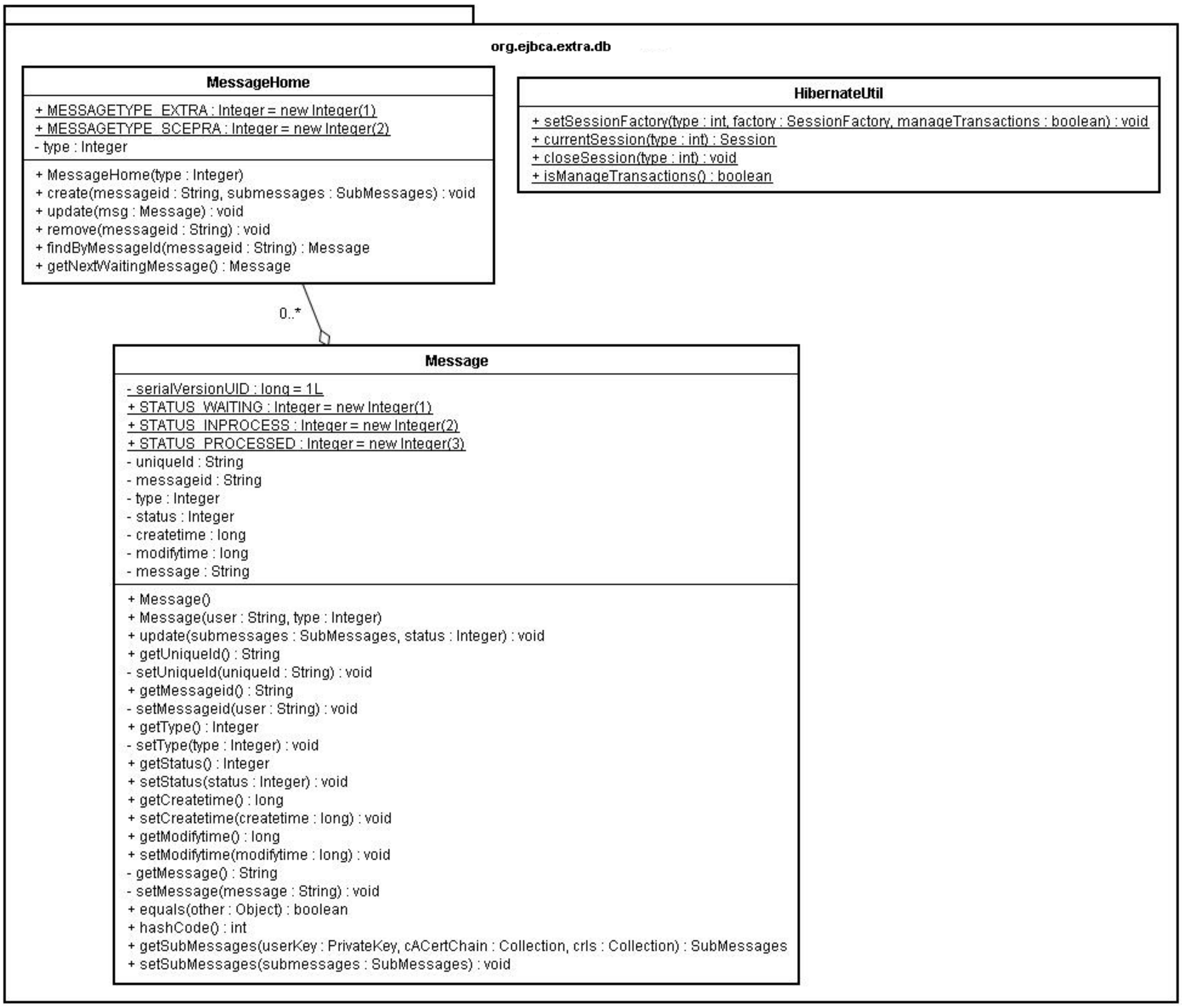

org.ejbca.extra.db
MessageHome
+ MESSAGETYPE_EXTRA : Integer = new Integer(1)
+ MESSAGETYPE_SCEPRA : Integer = new Integer(2)
- type : Integer
+ MessageHome(type : Integer)
+ create(messageid : String, submessages : SubMessages) : void
+ update(msg : Message) : void
+ remove(messageid : String) : void
+ findByMessageId(messageid : String) : Message
+ getNextWaitingMessage() : Message
HibernateUtil
+ setSessionFactory(type : int, factory : SessionFactory, manageTransactions : boolean) : void
+ currentSession(type : int) : Session
+ closeSession(type : int) : void
+ isManageTransactions() : boolean
0..*
Message
- serialVersionUID : long = 1L
+ STATUS_WAITING : Integer = new Integer(1)
+ STATUS_INPROCESS : Integer = new Integer(2)
+ STATUS_PROCESSED : Integer = new Integer(3)
- uniqueId : String
- messageid : String
- type : Integer
- status : Integer
- createtime : long
- modifytime : long
- message : String
+ Message()
+ Message(user : String, type : Integer)
+ update(submessages : SubMessages, status : Integer) : void
+ getUniqueId() : String
- setUniqueId(uniqueId : String) : void
+ getMessageid() : String
- setMessageid(user : String) : void
+ getType() : Integer
- setType(type : Integer) : void
+ getStatus() : Integer
+ setStatus(status : Integer) : void
+ getCreatetime() : long
+ setCreatetime(createtime : long) : void
+ getModifytime() : long
+ setModifytime(modifytime : long) : void
- getMessage() : String
- setMessage(message : String) : void
+ equals(other : Object) : boolean
+ hashCode() : int
+ getSubMessages(userKey : PrivateKey, cACertChain : Collection, crls : Collection) : SubMessages
+ setSubMessages(submessages : SubMessages) : void

Illustration 3: The Message DB Interface

4.1.1 MessageHome

In this class will the RA server mainly use two methods:

'create' that adds a message to the database, if a message already exists with the status 'waiting' or 'processed' it will overwrite the old message.

findByMessageId() will find a existing message given the unique messageId (could be username).

Both methods throws RuntimeException if database problems occur.

MessageHome will internally use the framework Hibernate to connect to the database.

4.1.2 Message

Class representing the columns in the message table.

Column	*Comment*
User	Unique messageId owning the request message, used as Primary Key, called User for historic reasons.
CreateTime	Time when the message was created
ModifyTime	Time when the message was last modified
SubMessage	String representation of a ExtRASubMessages
Status	Integer having the value, Waiting, Inprocess or Processed

The 'Message' table design in MySQL:

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| uniqueId   | varchar(250) | NO   | PRI |         |       |
| user       | varchar(255) | YES  |     | NULL    |       |
| type       | int(11)      | NO   |     |         |       |
| status     | int(11)      | NO   |     |         |       |
| createtime | bigint(20)   | NO   |     |         |       |
| modifytime | bigint(20)   | NO   |     |         |       |
| message    | longtext     | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

4.2 Request Classes

Basically there are four different requests: PKCS10 request assuming a certificate as response, PKCS12 and KeyRecovery requests assuming a PKCS12 key store in return and Revocation Requests used to revoke a generated certificate. Illustration 4 shows these classes and how they relate to each other. Illustration 5 shows more detailed content of these classes.

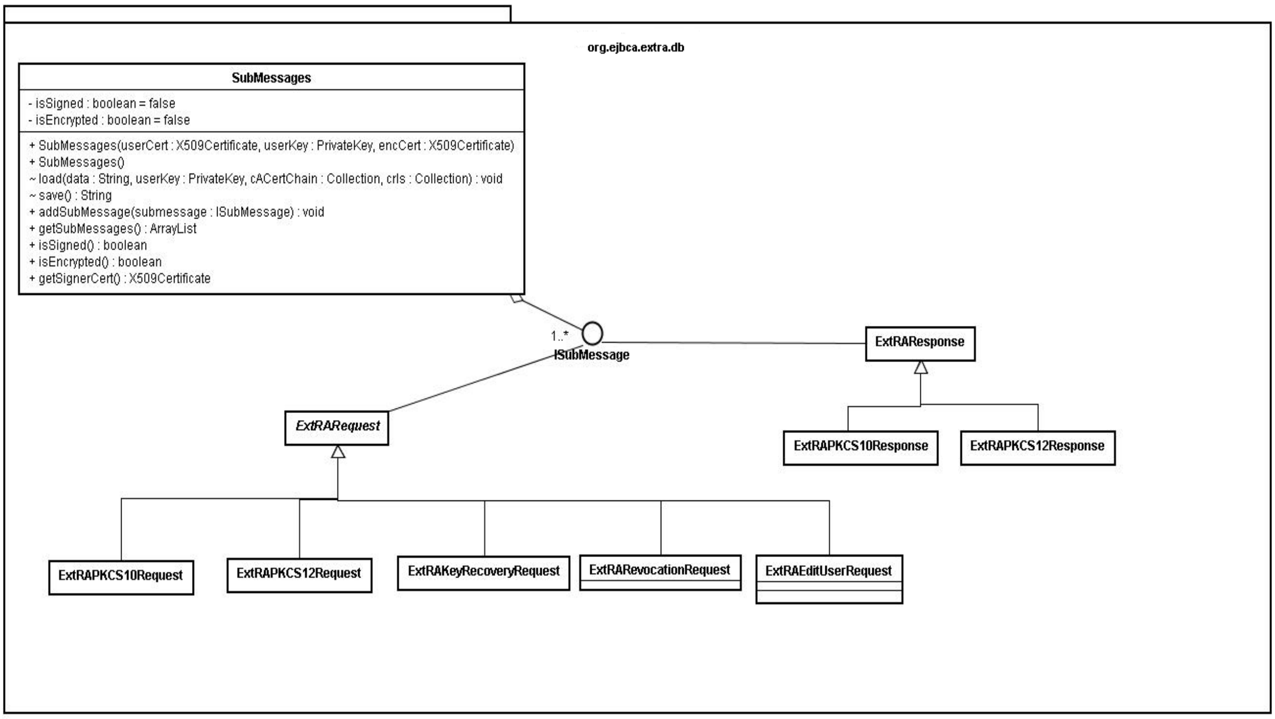

org.ejbca.extra.db
SubMessages
- isSigned : boolean = false
- isEncrypted : boolean = false
+ SubMessages(userCert : X509Certificate, userKey : PrivateKey, encCert : X509Certificate)
+ SubMessages()
~ load(data : String, userKey : PrivateKey, cACertChain : Collection, crls : Collection) : void
~ save() : String
+ addSubMessage(submessage : ISubMessage) : void
+ getSubMessages() : ArrayList
+ isSigned() : boolean
+ isEncrypted() : boolean
+ getSignerCert() : X509Certificate
1..*
ISubMessage
ExtRAResponse
ExtRAPKCS10Response
ExtRAPKCS12Response
ExtRARequest
ExtRAPKCS10Request
ExtRAPKCS12Request
ExtRAKeyRecoveryRequest
ExtRARevocationRequest
ExtRAEditUserRequest

Illustration 5: ExtRA Request Hierarchy

4.2.1 SubMessages

The SubMessages class i charge of collection sub-messages (request or responses) and serializing them into String format.

The submessage data can be signed and encrypted using CMS envelopes (PKCS7). For this it's required for both the CA Service and RA server have a PKCS12 keystore generated by a CA in the EJBCA instance. More information about the security is found in the Section 5.1.2.

4.2.2 ISubMessage

Empty interface that connects request and response messages.

4.2.3 ExtRARequest

Abstract base class containing request information about the user like dn, certificatetype ... Also contains a requestId used to match the request with the given response.

4.2.4 ExtRAPKCS10Request

Class containing a PKCS10 request.

4.2.5 ExtRAPKCS12Request

Class containing a password used to lock the key-store, a key size and key algorithm.

4.2.6 ExtRAEditUserRequest

Class containing userdata used to add or edit a user. Mostly used with hard token issuing.

4.2.7 ExtRAKeyRecoveryRequest

Class containing serial number of certificate with associated key to recover. It is possible to request a new certificate with the original keys, or the original certificate, depending on the constructor used.

4.2.8 ExtRARevocationRequest

Request to use to revoke a generate certificate. Contains the IssuerDN, certificate SN and revocation reason (One of the ExtRARevocationRequest.REVOKATION_REASON_ constants). Optionally you can request revocation of the user in EJBCA, so the user can not get a new certificate, when revoking the user, all the users certificates are revoked. This is requested by setting the parameter *revokeuser* to true. You can also optionally request revocation of all the users certificates, but without revoking the user itself, do this by setting *revokall* to true.

4.2.9 ExtRAResponse

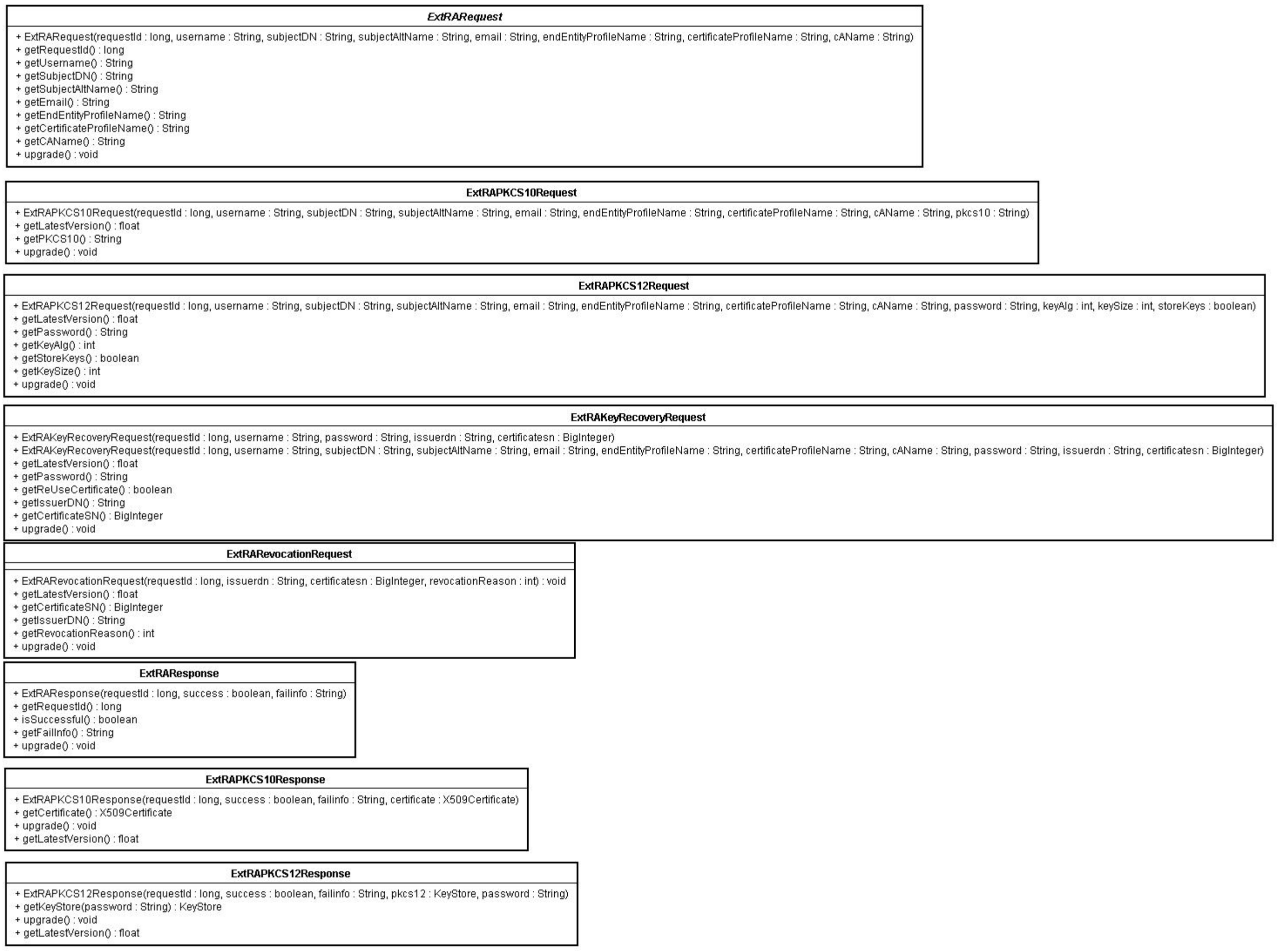
Base class for all ext RA responses. Contains the requestId, a flag if operation was successful and a failure message if not. A basic ExtRAResponse will be returned from a ExtRARevokationRequest.

4.2.10 ExtRAPKCS10Response

Response to a ExtRAPKCS10Request, contains a certificate if operation was successful.

4.2.11 ExtRAPKCS12Response

Response to a ExtRAPKCS12Request and ExtRAKeyRecoveryRequest, contains a Java Key Store of type PKCS12 containing the certificate and private key if the operation was successful.



```
ExtRARequest
+ ExtRARequest(requestId : long, username : String, subjectDN : String, subjectAltName : String, email : String, endEntityProfileName : String, certificateProfileName : String, cAName : String)
+ getRequestId() : long
+ getUsername() : String
+ getSubjectDN() : String
+ getSubjectAltName() : String
+ getEmail() : String
+ getEndEntityProfileName() : String
+ getCertificateProfileName() : String
+ getCAName() : String
+ upgrade() : void

ExtRAPKCS10Request
+ ExtRAPKCS10Request(requestId : long, username : String, subjectDN : String, subjectAltName : String, email : String, endEntityProfileName : String, certificateProfileName : String, cAName : String, pkcs10 : String)
+ getLatestVersion() : float
+ getPKCS10() : String
+ upgrade() : void

ExtRAPKCS12Request
+ ExtRAPKCS12Request(requestId : long, username : String, subjectDN : String, subjectAltName : String, email : String, endEntityProfileName : String, certificateProfileName : String, cAName : String, password : String, keyAlg : int, keySize : int, storeKeys : boolean)
+ getLatestVersion() : float
+ getPassword() : String
+ getKeyAlg() : int
+ getStoreKeys() : boolean
+ getKeySize() : int
+ upgrade() : void

ExtRAKeyRecoveryRequest
+ ExtRAKeyRecoveryRequest(requestId : long, username : String, password : String, issuerdn : String, certificatesn : BigInteger)
+ ExtRAKeyRecoveryRequest(requestId : long, username : String, subjectDN : String, subjectAltName : String, email : String, endEntityProfileName : String, certificateProfileName : String, cAName : String, password : String, issuerdn : String, certificatesn : BigInteger)
+ getLatestVersion() : float
+ getPassword() : String
+ getReUseCertificate() : boolean
+ getIssuerDN() : String
+ getCertificateSN() : BigInteger
+ upgrade() : void

ExtRARevocationRequest
+ ExtRARevocationRequest(requestId : long, issuerdn : String, certificatesn : BigInteger, revocationReason : int) : void
+ getLatestVersion() : float
+ getCertificateSN() : BigInteger
+ getIssuerDN() : String
+ getRevocationReason() : int
+ upgrade() : void

ExtRAResponse
+ ExtRAResponse(requestId : long, success : boolean, failinfo : String)
+ getRequestId() : long
+ isSuccessful() : boolean
+ getFailInfo() : String
+ upgrade() : void

ExtRAPKCS10Response
+ ExtRAPKCS10Response(requestId : long, success : boolean, failinfo : String, certificate : X509Certificate)
+ getCertificate() : X509Certificate
+ upgrade() : void
+ getLatestVersion() : float

ExtRAPKCS12Response
+ ExtRAPKCS12Response(requestId : long, success : boolean, failinfo : String, pkcs12 : KeyStore, password : String)
+ getKeyStore(password : String) : KeyStore
+ upgrade() : void
+ getLatestVersion() : float
```

Illustration 6: Class diagram of RA Request and Responses

5 Using the Ext RA API

5.1 Configuring the Ext RA CAService

The first step is to do before installing using the Ext RA API is to install the Ext RA CAService in EJBCA.

Step 1. If the RA messages should be signed and/or encrypted then two key stores need to be generated using one of the CAs i the EJBCA installation,one for the RA server and one for the RA-CAService. In the examples we use AdminCA1 to issue these keystores. If encryption should be used, must the RA-CAService keystore's certificate also be downloaded separately, to be used by the RA server. This can be done in the page 'View Certificate' in the adminweb. *Important*, it must be the same CA issuing the RA and CAService keystores.

The RA-CAServer keystore must be in named `extrakeystore.p12` and be in the keystore subdirectory. The RA server key store can have any name.

Step 2. Configure the `RAMessageDS` data source in JBoss by editing the file src/appserver/jboss/ramessage-ds.xml.

Step 3. Configure the RACAService by editing the file src/appserver/jboss/extraca-service.xml. The following options should be configured:
`Polltime` (5SEC used during tests)

```
EncryptionRequired (true|false)
SignatureRequired (recommended)
KeyStorePassword
RAIssuer (CA Name of the CA issuing RA Certificates, used to check the validity
of RA signatures.)
```

Step 4. Deploy the RA-CAService keystore by using: `ant deploy-keystore`

Step 5. Make sure a mysql connector is installed in Jboss. It should be in the JBOSS_HOME/server/default/lib directory.

Step 6. Build and deploy the RA CAService using the command: `ant deploy-ra-caservice`
This will compile and copy the service to JBoss. Make sure you have set the environment variables EJBCA_HOME and JBOSS_HOME first.

Important, the RA CAService is dependant on a new jar in EJBCA called ejbca-util.jar if your current version of EJBCA doesn't create this jar (in EJBCA_HOME/dist) upgrade the EJBCA distribution first.

Step 7. Check the server logs that everything seems fine by checking file JBOSS_HOME/server/default/log/server.log

5.1.1 *Security options*

It is strongly recommended to at least use signing of messages sent between RA and CA. If the messages are signed will the RAs certificate be used for authorization internally. This makes it possible to trace which RA that approved the information certified and possible to control which kind of information that the RA can approve, by defining End Entity Profiles.

For signing is the SHA-256 digest algorithm used and for encryption is AES256 used.

5.1.2 *A word about Authorization*

If message signing is used, must the RA servers certificate (used to sign the message) be an administrator i EJBCA.

The administrator must have access rights to the following´:

1. Have the administrator flag marked in the user data.
2. Belong to and administrator group with the following access rules
   - At least have the role as RA administrator.
   - View/Create/Edit/Key Recovery/Revocation Rights.
   - Access to the End Entity Profiles used by the RA.
   - Access to the CAs used.

If data sent isn't signed will the service run as an 'internal user' where anything goes (Super Administrator).

5.2 *Using the RA API*

After running ant is the API libraries generated in the directory `dist/client-jars` it contains all the jars necessary to use the API.

5.3 *Configuring Hibernate*

Before using the API Hibernate must be configured in the code. This can be done by a hibernate configuration file or directly in the code. Example:

```
   static {
        CertTools.installBCProvider();
        Properties props = new Properties();
        try {
props.load(ExtRATestClient.class.getResourceAsStream("/log4j.properties"));
                       PropertyConfigurator.configure(props);

               } catch (IOException e) {
                       e.printStackTrace();
               }
        Configuration dbconfig = new Configuration().
            setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect").
            setProperty("hibernate.connection.driver_class", "com.mysql.jdbc.Driver").
            setProperty("hibernate.connection.url", "jdbc:mysql://localhost/messages").
            setProperty("hibernate.connection.username", "test").
            setProperty("hibernate.connection.password", "foo123").
            setProperty("hibernate.connection.autocommit", "true").
            setProperty("hibernate.cache.provider_class",
"org.hibernate.cache.HashtableCacheProvider").
            setProperty("hibernate.hbm2ddl.auto", "update").
           // setProperty("hibernate.show_sql", "true")
            addInputStream(ExtRATestClient.class.getResourceAsStream("/Message.hbm.xml"));


            HibernateUtil.setSessionFactory(HibernateUtil.SESSIONFACTORY_RAMESSAGE,
dbconfig.buildSessionFactory(), true);


    }
```

A hibernate configuration file could look like:

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory name="java:/registrationDatabase">
                <property name="show_sql">false</property>
                <property name="connection.datasource">java:/hibernateDatasource</property>
                <property name="hbm2ddl.auto">update</property>
                <property
name="cache.provider_class">org.hibernate.cache.HashtableCacheProvider</property>
                <property name="transaction.flush_before_completion">true</property>
                <property name="connection.release_mode">after_statement</property>
                <property
name="transaction.manager_lookup_class">org.hibernate.transaction.JBossTransactionManagerLookup</pro
perty>
                <property
name="transaction.factory_class">org.hibernate.transaction.JTATransactionFactory</property>
                <mapping class="org.ejbca.extra.ui.admin.db.RegistrationData"/>
                <mapping resource="Message.hbm.xml"/>
        </session-factory>
</hibernate-configuration>
```

If combined with a Jboss datasource:

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
    <local-tx-datasource>
        <jndi-name>hibernateDatasource</jndi-name>
        <connection-url>jdbc:mysql://127.0.0.1:3306/messages?characterEncoding=UTF-8</connection-
url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>test</user-name>
        <password>foo123</password>

    </local-tx-datasource>
</datasources>
```

5.4 Using Signing and Encryption of Messages

If signing or encryption of messages is used, then should the RA have a key store and the RA CAService certificate. These are used in the constructor

```
SubMessages(X509Certificate userCert, PrivateKey userKey, X509Certificate encCert)
```

and the method in Message

```
getSubMessages( PrivateKey userKey, Collection cACertChain, Collection crls)
```

To disable CRL checking send null as parameter. CACertChain is the CA chain that signed the RA and CAService keystore.

If no security should be applied to the messages use 'null' for all the parameters.

5.5 API-Examples

This section provides a couple of examples of how the API could work from the RA server developers point of view.

More information about using the API can be found in the junit scripts and the test client se/primeKey/rasrv/ra/ExtRATestClient.java

5.5.1 Example 1: Adding a new request

```
MessageHome msgHome = new MessageHome(MessageHome.MESSAGETYPE_EXTRA);

SubMessages subMessages = new SubMessages();

subMessages.addSubMessage(new ExtRAPKCS10Request(requestId1, username, subjectDN,
subjectAltName, email, endEntityProfileName, certificateProfileName, cANAme, pkcs10));

subMessages.addSubMessage(new ExtRAPKCS12Request(requestId2, username, subjectDN, subjectAltName,
email, endEntityProfileName, certificateProfileName, cANAme, password,
ExtRAPKCS12Request.KEYALG_RSA, 1024, storeKeys));

try {
  msgHome.create(messageId, subMessages);
}catch(RuntimeException e){
  // Problems with database
}
```

5.5.2 Example 2: Checking if a request is ready

```
Try{

  Message msg = msgHome.findByMessageId(messageId);
  if(msg != null){
    if(msg.getStatus == Message.STATUS_PROCESSED){
       // Message is ready to use. In  example 5.1 we requested a pkcs10 and a pkcs12 so that
       // what we will get in the response if all i ok
       Iterator iter = msg.getSubMessages().getSubMessages().iterator;
       while(iter.hasNext()){
          ExtRAResponse resp = (ExtRAResponse) iter.next();
          if(!resp.getSuccessful()){
             // Request Failed report using resp.getFailInfo();
          }
          if(resp instanceof ExtRAPKCS10Response){
            req1cert = ((ExtRAPKCS10Response) resp).getCertificate();
          }
          if(resp instanceof ExtRAPKCS12Response){
            req1keystore = ((ExtRAPKCS12Response) resp).getPKCS12();
          }
       }

    }else{
      // Respons with still pending.
    }
  }else{
    // Respond with user doesn't exist.
  }
}catch(RuntimeException e){
  // Problems with database
}
```

6 Testing

The following tests have been done.

6.1 Automatic Junit Tests

Automatic Junit tests lies in the directory '`src/tests`'

The following must be done before running the tests.

1. Setup a mysql database on localhost and create a 'messages' database and user 'test' with password 'foo123'.
2. Deploy the RA CA Service without any requirements on signature and encryption.
3. Enable Key Recovery in EJBCA System Configuration.

The following tests are performed:
TestExtRAMsgHelper: Performs test related to encryption/signature of messages.
TestExtRAMessages: Makes sure that request and response classes are serialized properly
TestMessageHome: Makes basic database functionality tests.
TestRAAPI: Makes a full scale tests of sending PKCS10 and PKCS12 request to the CA and waits for proper responses. May take some time and check the server log for errors. It also test to revoke some of the generated certificates.

6.2 Ext RA Test Client

To simply load and performance tests have a test client been implemented that generates request from x concurrent RAs. Using this client it's possible to figure out expected response times for the different types of requests.

The test client is in the jar dist/client-jars/ext-raclient.jar and executed by the command.
java -jar dist/client-jars/ext-raclient.jar.

The following parameters can be set:

```
<CERT | KEYSTORE> : Type of test, CERT creates single PKCS10 requests, KEYSTORE creates one PKCS10
                    and one PKCS12 request for each message
<dbhost>          : Hostname of database.
<KeyStorePath>    : The path to the keystore used to sign/encrypt messages. Use NOKEYSTORE for
                    unencrypted security level.
<KeyStorePwd>     : Password to unlock the keystore,. Use NOPWD for unencrypted security level.
<EncCert>         : Path to certificate (DER) used to encrypt messages,. Use NOCERT for unencrypted
                    security level.
<SecurityLevel>   : Security Level, Valid values are UNSECURED, SIGNED, ENCRYPTED, SIGNEDENCRYPTED
<RequestsPerMin>  : Requests to generate every minute per concurrent RA.
<ConcurrentRAs>   : Number of concurrent RAs that will create requests.
<WaitTime>        : Number of seconds to wait for answer before exception is thrown.

Examples : Simple test sending unsecured requests every 5 s in one thread  expecting an answer
within 60 s :
java -jar ext-raclient.jar CERT NOKEYSTORE NOPWD NOCERT UNSECURE 12 1 60
```

```
Advanced test using encrypted and signed requests for pkcs10 cert and a pkcs12 keystore
every 1 min in two threads, expecting an answer within 60 s :");
 java -jar ext-raclient.jar KEYSTORE rakeystore.p12 foo123 enccert.cer SIGNEDENCRYPTED 1 2 60
```

Important, if signing is used must the RA server key store be configured as an Super Administrator in EJBCA.

The test client have the same requirements on the database as the junit tests.

6.3 Manual Tests

The following test have been done verified.

1. RA and CAService keystores generated by a Root CA.
2. RA and CAService keystores generated by a SubCA.
3. Unsigned messages isn't accepted if signing is set as required.
4. Unencrypted messages isn't accepted if encryption is set as required.
5. If RA certificate is revoked the wont any more messages be accepted.
6. If CA certificates (Either Root or Sub) is revoked wont the any more message be accepted
7. Check that if all certificates are revoked, then is also the user status set to 'Revoked'.
8. Authorization tests of signed messages:
    - The administrator flag isn't set
    - The administrator haven't access to the CA specified in request.
    - The administrator haven't access to the End Entity Profile specified in request.
    - The administrator haven't create rights
    - The administrator haven't got edit rights
    - The administrator haven't got key recovery rights

7 References

Hibernite, http://www.hibernate.org/