Bachelor Thesis

Fall 2012

School of Health and Society
Department Design and Computer Science

License Management for EBITool

Author
Anton Krznaric

Instructor
Henrik Jönsson
Andreas Nilsson

Examiner
Christian Andersson

Document Page

School of Health and Society
Department Design and Computer Science
Kristianstad University
SE-291 88 Kristianstad
Sweden

Author, Program and Year:
Anton Krznaric, Computer Science 2012

Instructor:
Henrik Jönsson, B. Sc., Bombardier Transportation
Andreas Nilsson, M. Sc., HKr

Examination:
This graduation work on 15 higher education credits is a part of the requirements for a *Degree of Bachelor in Computer Science* (as specified in the English translation).

Title:
License Management for EBITool

Abstract:
This degree project deals with license management for EBITool. It´s about providing protection and monitoring for a Java Application via a license server, and the construction of it. An analysis that discusses the approach and other possible courses of action is also included. Additionally, it covers a discussion of a prototype implementation of the model solution from the analysis.

The prototype is a Java EE application that deploys to JBoss AS7. It´s developed using the JBoss Developer Studio 5.0.0, an Eclipse IDE with JBoss Tools preinstalled. It exposes web services to Java Applications through SOAP via JAX-WS. Using Hibernate, the web service Enterprise Java Beans get access to a PostgreSQL 9.1 database via entity classes mapped to the database through the Java Persistence API.

Language:
English

Approved by:

_____

Prof. Christian Andersson Date
Examiner

Table of Contents

1 Introduction

1.1 Context

EBITool is a tool suite that is based on Eclipse with plugins providing specific functionalities. It has currently neither monitoring nor protection of its usage. This thesis intends to suggest a solution for this that can be used by the tool suite.

The tool suite should be license controlled and monitored by connecting to a license server application, preferably a Java EE [1] implementation. There should also be a way to create licenses and monitor their usage. The monitoring and license creation can be provided by a simple Java application using JAX-WS [2] with a CLI (Command-Line Interface), taking in mind that further work might including transforming this into a web interface making the administration possible without the need of a client java application to the benefit of a web interface.

1.2 Aim and Purpose

The purpose of this project is to analyze different approaches and provide a solution that takes form in a prototype that can easily be applicable on the tool suite.

The aim includes creating a secure license server application that is deployable to a full or partial implementation of Java EE, i.e. GlassFish [3], JBoss AS [4], Tomcat [5] etc. It should be flexible, portable, implement security to a level so that the license server doesn´t accept any license that is not valid or provided by it. Furthermore, it should be taken into consideration that it will later be implemented in the EBITool tool suite.

1.3 Method and Resources

The license management is to be handled by a server application which monitors the usage of the application. A Java desktop application shall be able to be license managed and the usage monitored.

For this the Java EE 6 platform is used. The specific implementation for the prototype is JBoss AS7 that runs on a development computer. As a database on the server-side PostgreSQL 9.1 [6] is used. For the development of the prototype, JBoss Developer Studio 5.0.0 [7] is used. During development all applications run on a Windows 7 [8] 64-bit PC.

1.4 Presentation of the Company

Bombardier Transportation is a company that among other provides solutions in train technology. This includes signaling systems for controlling train movement.

The company delivers complete main line systems, sub systems and individual products to railway operators and infrastructure owners across the globe and is among the world´s largest companies in the world in the rail-equipment manufacturing and servicing industry.

As at December 31, 2011, the total workforce of Bombardier Transportation across the globe extends to 36,200 employees [9]. The company holds 62 production and engineering sites, 18 service centers in 26 countries worldwide.

The Bombardier Transportation official web site is available at http://www.bombardier.com/en/transportation (Last access: January 15, 2013).

1.5 Acknowledgements

I would like to dedicate this part to thank the people around me that have enlightened me with positive inspiration and momentum simplifying my efforts in performing the task of completing this Bachelor Thesis.

Thank You.

2 Analysis

2.1 Information Retrieval

The requirements and purpose of the thesis has been retrieved by meetings with the instructor at Bombardier Transportation, Henrik Jönsson, and by an experienced developer from the EBITool developers' team at Bombardier Transportation, Fredrik Salomonsson, who has advised from his experience in cryptography and Java EE solutions.

The cryptographic solution and the server-client approach I have developed based on my knowledge and experience on the subject and using the Java API [10], scanning the web and books for information. Links to interesting web resources and books are presented in the references chapter.

2.2 Theory Model

To keep an application under license control, the licensee must be provided with some kind of key to be able to verify its access through the license managed application. This key may not be transferred over the network since it should remain secret to everyone except the specific licensee. By using public-private key cryptography this key doesn't even have to be known by the license server, which improves the licensee integrity in case of a security breach. In Figure 1, it´s shown between which components this access verification is performed.

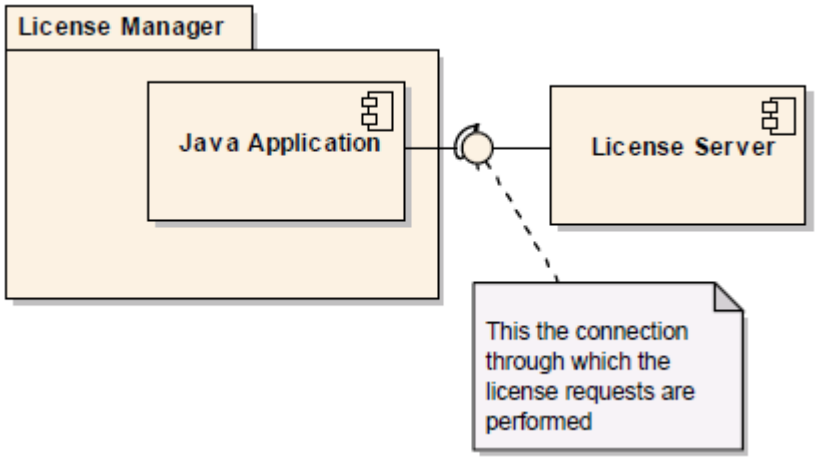


Figure 1: Illustrating the connection through which the license requests are performed

To identify a licensee, each license has a unique ID along with the key. To prove that the licensee has the key, it transfers its unique ID to the license server, and get as response random generated data. This data is stored by the license server in a database, associated with the licensee. When the licensee receives the random generated data, it can sign it with its key, which is the private key of a public-private key pair, of which the license server has the public key associated with this private key, associated with the licensee. This random generated data is only used for one request by the licensee and is after that removed and for next request new random data must be generated, so called nonce [11].

When the licensee has signed the nonce with its private key, it sends the signed nonce back to the license server in serialized form. The license server can now verify the identity of the licensee and be sure that it has the private key of the key pair associated with it.

If the licensee´s license is valid, i.e. it has not expired and the signed content is verified against the public key of the licensee, an access token is returned to the license managed application. Otherwise an invalid access token is returned, indicating that the license is invalid.

The access token contains the unique id of the licensee along with a String array containing explicit access of specified components. A String in the array expresses access to the certain component identified by that String.

This procedure is carried out iteratively with an interval specified in the license managed application. By setting this interval to zero (0) it´s indicated that only one (1) initial request will be made.

For each valid request by the license managed application an entry is saved in the database, logging the usage of the license managed application. This entry contains the unique id of the licensee along with the time of the request.

The admin server works similarly to the license server except that it accepts only one licensee, the admin licensee. If the request of the license admin client responds to the unique id of the admin licensee, a nonce will be returned to the license admin client for signing. If the signature is verified against the public key of the admin licensee, the licensee that the admin licensee has requested to add is stored in the database after which the credentials are returned to the admin client.

The credentials consist of the unique id of the newly created licensee and an encrypted session key with which the private key is encrypted that also follows. The session key is a randomly generated secret key for each request and encrypted with the public key of the admin licensee making sure only the admin licensee can decrypt the private key of the new licensee from the credentials.

2.3 Alternative Models/solution

The public-private key pair solution above using verification of signed randomly generated data to prove the integrity of the licensees is a straight-forward approach proved to work by cryptography.

As an alternative model for providing the best usage control and monitoring, actually the most efficient one, is to run the whole software remotely against a controlled central server, which would give the owner full control of its software, i.e. not distribute software that should be controlled at all. This is shown in Note 1 of Figure 2 below, which illustrates this approach.

For an application to implement license management, no server connection needs to be maintained, but since this license management model needs to implement monitoring, a

persistent server connection has to be maintained. When verifying that the licensee has the key that proves it has the license, the key should not be transferred over the network, since it could be retrieved by a MITM [12] attack or similar. Instead, it has to be encrypted somehow, or in another manner proved to be in possession. Nevertheless, it should not be encrypted the same way each time, since it would result in the encrypted key being the actual key, hence the purpose of encryption is wasted.

Also, when creating the license, the credentials that are returned to the administrator, has to be encrypted. If they are not encrypted, the credentials can be caught by anyone listening on the network.

Figure 2 below shows three (3) different approaches of providing license management to an application. A note is attached to each approach explaining what is visible to the client application. Connected to Note 2, the approach described in the Theory Model section is compared to the other approaches.

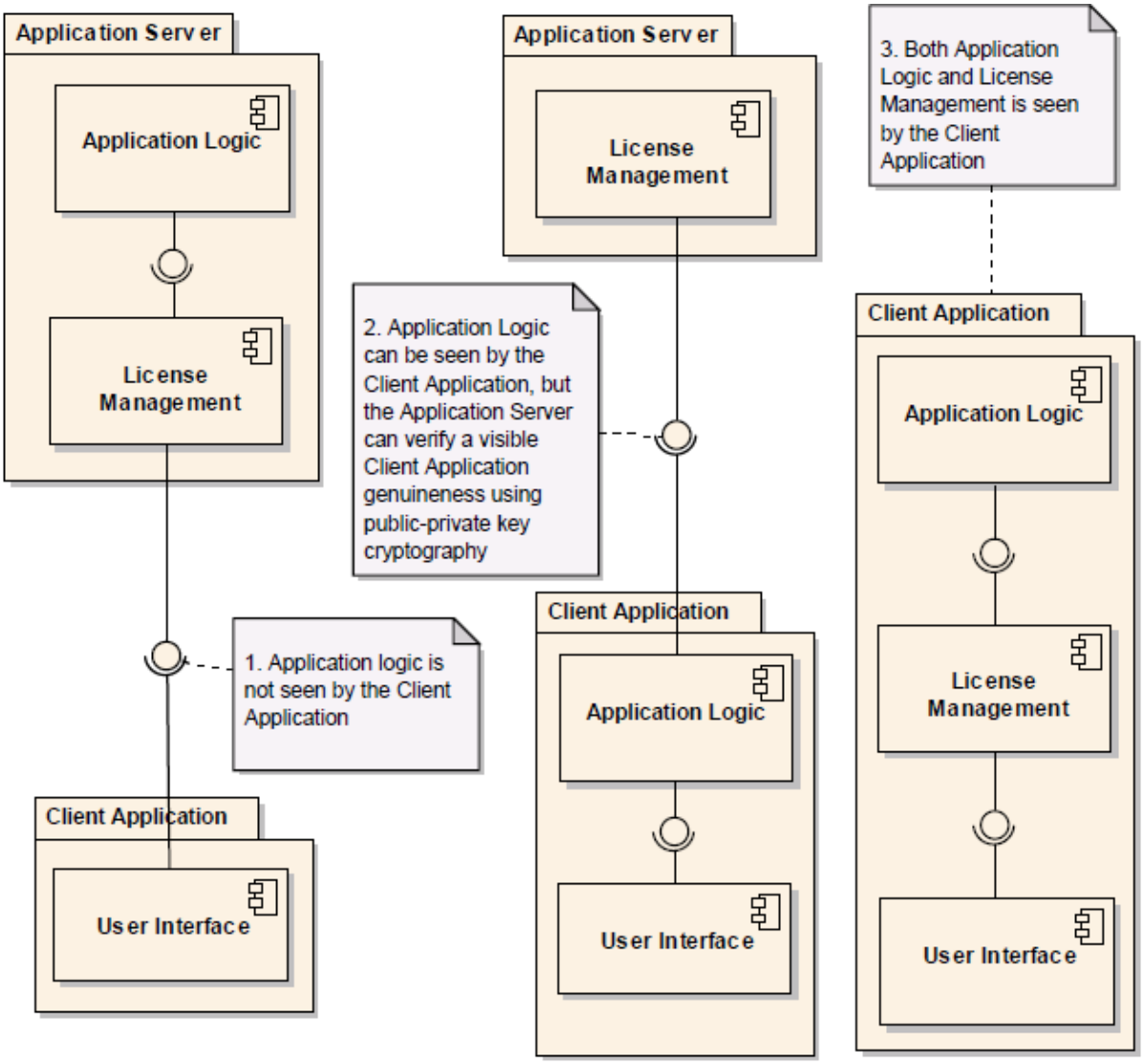


Figure 2: Illustrating different approaches of providing license management to an application

2.3.1 Existing solutions

Most of the software existing providing similar functionality is proprietary and closed-source. They also usually have inadequate documentation of the security and encryption procedures. Common among all these solution are that they need to have a key, a serial number or similar. They would also need to verify this key against a server without revealing it. In some implementations, it´s mistakenly believed that these keys can be encrypted and therefore secure. This is only partly true. If a static key is encrypted with another static key, i.e. if one specific key encrypts the same key every time, the result is that the encrypted version of the key becomes the key, hence the encryption is useless.

Other solutions, like TrueLicense [13] provides open source license management for closed source applications. The purpose of this solution is somewhat different than the purpose of the approach in the solution provided in this document. The TrueLicense doesn´t provide monitoring against a license server when running the client application, which makes the security considerations and goals of the use totally different.

Since the main purpose of the TrueLicense project is to make a Java application license controlled, i.e. you´ll need a key or a serial number to run the application, without using a license server, it´s not even taking into consideration how monitoring of the application usage would be performed. Instead, more effort has been made around the security on the local (uncontrolled) computer (which is really impossible), introducing terms like code scrambling and similar.

The TrueLicense license management approach is illustrated in Figure 2 above, under Note 3. Here it is shown how all logic is visible to the client application.

2.4 Environmental Consequences

The approach of having a license managed application is dependent on an internet connection. The consequences of server down-time, internet connection failure where the application runs can be severe, since it might not allow the application to run. This can suppress work time, causing the application to be temporarily useless. Furthermore, if the application otherwise is dependent on an internet connection and already is dependent on servers being up, this doesn’t have to be seen as a remarkable drawback.

3 Realization

3.1 Design

The server application handles the verification of licensees. There are two types of licensees: the admin licensee and the application licensee. The admin licensee is used by a client application that administrates and monitors the licensees. The application licensee is used by a client application to request access to its content and to be administrated by the admin licensee. The communication with the server is dealt by through the usage of web services.

The server application exposes two web services: AdminServiceWS and LicenseServerWS. The first is used by the admin licensee and the second is used by the application licensees.

The administration of licensees and monitoring of their respective application usage is provided by the web service called AdminServiceWS. This web service exposes an EJB [14] called AdminServiceBean through an interface called AdminService, which is accessed through SOAP [15].

The client application that uses the AdminServiceWS is called LicenseAdminClient. It contains the JAX-WS Runtime to be able to communicate with the web service. The client needs a license file with admin credentials. The file consists of two properties: the admin UUID [16] and the admin private key. The admin UUID is used as a unique identifier by the client application to identify the admin to the AdminServiceWS, and the admin private key is used to verify its authenticity. The bytes of the String representation of the UUID are stored in the license followed by the encoded form of the private key. The string representation of the UUID is described by a BNF in the JavaDoc for java.util.UUID.toString() [17]. The admin private key is the private key of a 2048-bit RSA public-private key pair [18] randomly generated and associated to the admin UUID.

The license managed application itself is extended with a .jar-file containing the application that communicates with LicenseServiceWS. This application contains JAX-WS Runtime to be able to communicate with the web service. The application that is put under license control also has a license file, containing the credentials of the licensee. These credentials are the same as the one used by the LicenseAdminClient, although these are associated to this specific licensee. The file location is specified in the application and is handed over to the LicenseManagedApplication interface which is implemented by request. The thread that handles the request is started in the class that implements the LicenseManagedApplication interface and is called LicenseManager.

The nonce that is used to verify the integrity of the licensees is an SHA-1 hash [19] (160-bit message digest) of the String representation of a randomly generated Integer.

3.1.1 Components

The component diagram in Appendix C explains how the different components interact with each other.

The main components, which are the web services in the server application, communicate with each other indirectly via the PostgreSQL database. The database is connected to the application server as a data source through Hibernate [20]. The data source, in turn, is mapped to entity classes in the server application through JPA [21], Java Persistence API, as shown in Figure 3 below.

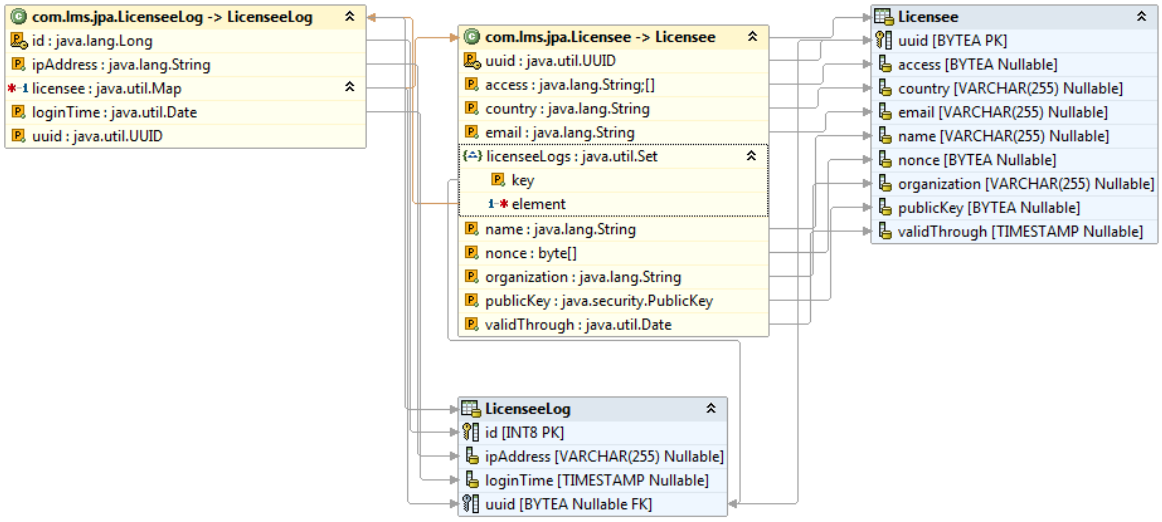


Figure 3: mapping diagram illustrating the mapping of the entity classes to the tables in the database

The server application is a Java EE 6 enterprise application project running on a JBoss AS7. The class diagram in Appendix B illustrates the how the classes of this server application depends on each other, their members and methods, as well as their relation to each other and which Java Annotations the classes declare. The client applications are Java SE 7 [22] CLI applications, to which JAX-WS Runtime is added along with the client project of the enterprise application project that contains classes and interfaces that are exposed and shared. These applications are split up in projects, of which some depends on others, which is clarified in the table of Appendix A, describing all development projects.

When the AdminServiceBean creates a license, it instantiates the Licensee class, which is a JPA Entity, and persists the instance to the data source. The AdminServiceBean contains the business logic of the AdminServiceWS web service. Later, when the LicenseServiceBean gets an access request, it finds the Licensee instance via JPA from the passed on UUID and can get the data for that licensee, such as public key, access parameters and so forth. This also applies for the AdminServiceBean upon getLicenseesXML operation.

For the LicenseeLog entity class it´s somewhat contrary – the LicenseServiceBean stores data upon access request operations from Licensees – and the AdminServiceBean retrieves the data upon getLicenseesXML operations.

The LicenseServiceBean and the AdminServiceBean are the business logic implementation of the web services LicenseServiceWS and AdminServiceWS respectively, exposing the LicenseService and AdminService interfaces through JAX-WS.

The LicenseManager component is the application that is implemented in the application that gets license managed. This component handles all communication with the LicenseServiceWS web service and calls methods on the license managed application upon failure and sets fields based on explicit access parameters retrieved from access requests. This is made possible by letting the license managed application implementing the LicenseManagedApplication interface.

3.1.2 Security

Since it´s required for the server application to be sure the licensee that requests access is really the one in possession of the license file – the file that is associated with the application licensee, containing the licensee UUID and the licensee private key – the key must somehow be verified. This is realized by signing randomly generated data that is used for one and only one request. When this request is done, the nonce that has been temporarily saved to the database is erased. For the next request, a new nonce must be created, stored in the database, and sent to the client application for signing. The access token that is returned to the license manager on the client side includes no sensitive data that can be used to alternate the integrity of the license which is why this is sent unencrypted.

The license credentials that the license admin client receives include sensitive data. It´s of importance that the private key of these credentials remain secret and is not transferred unencrypted over the network. Because of this the key is encrypted with a session key, that in turn is encrypted with the public key of the admin licensee of which the private key the admin licensee is in possession of.

3.1.3 Encryption

The encryption of the credentials works like this:

For better overview, the nonce steps are also included

1. An admin RSA 2048-bit key-pair exists, of which the license server application has the public key and the license admin client has the private key.

2. When the license admin client wants to create a license, it first asks for a nonce. The license server application generates the nonce by generating a random integer and hashes it to a 160-bit message digest with SHA-1.

3. The bytes of the nonce are stored in the nonce field of the admin licensee in the database, and returned to the license admin client.

4. The license admin client uses its private key to sign the nonce using a java.security.SignedObject. It serializes the object and sends the bytes to the license server application when requesting an access token.

5. The license server application retrieves the serialized signed nonce, deserializes it, and verifies the signature with the public key of the admin licensee from the database. This way it can be sure that the request is made by the admin licensee, i.e. the one in possession of the private key of the admin license.

6. Now the license admin client sends a request to the license server application to create a license and passes parameters to the new license in the method invocation.

7. The license server application creates a license, with the requested parameters, and generates a unique id, a UUID, for the new license. It also generates a public-private RSA 2048-bit key-pair for the new license. The public key is stored with the parameters in the database, and the private key is encrypted with a generated AES 256-bit key that is also generated for this request. The AES 256-bit key is encrypted with the public key of the admin licensee and returned to the license admin client along with the encrypted private key and the unique id of the new licensee, all in an instance of the Credentials class.

8. When the license admin client receives the credentials, it decrypts the AES 256-bit key, and uses this key to decrypt the private key. This private key is stored along with the unique id of the credentials to a license file which then can be used as a license with a license managed application, i.e. a java application implementing the LicenseManagedApplication interface running the LicenseManager.

These steps are performed by the AdminService component of the license server application, using to methods of the interface AdminService, getNonce() and createLicense(). The LicenseManager is using the CryptoUtils class of the lms-utils project as a utility to decrypt the keys. The sequence in which these steps are used is further explored in the next section about functioning along with an illustration of the sequence, see Figure 3.

3.2 Functioning

Upon start and repeated at a given interval the LicenseManagedApplication, through the LicenseManager thread, performs the following request to the LicenseServiceWS, which is realized in an object of the type LicenseService through JAX-WS:

1. Connect to the LicenseServiceWS web service by service name and the URL of the WSDL document of the web service, and retrieve a LicenseService object.

2. Perform a request access operation.

3. Start a LicenseManager thread that continuously performs a request access operation on a given interval.

The sequence diagram in Figure 4 illustrates the program flow during this operation.

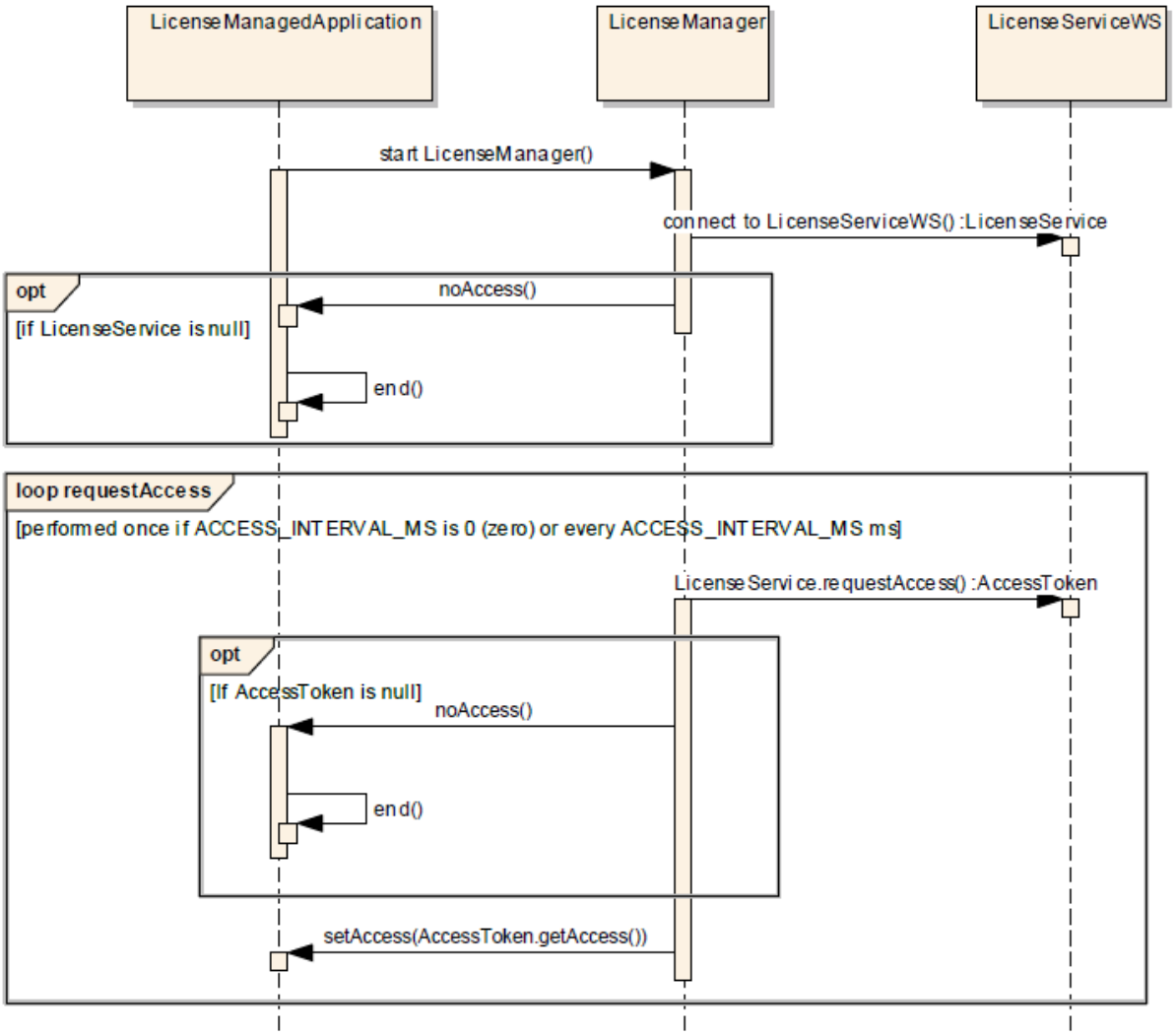


Figure 4: sequence diagram of the request access operation

The LicenseAdminClient is used to create licenses and to retrieve a log of each licensee´s requests. When starting the LicenseAdminClient it connects to the AdminServiceWS web service by service name and the URL of the WSDL document of the web service, and retrieves an AdminService object.

When creating a license the following operations are performed:

1. Perform a create license operation.

2. Upon success, store the credentials to a file.

The sequence diagram in Figure 5 illustrates the program flow during this operation.

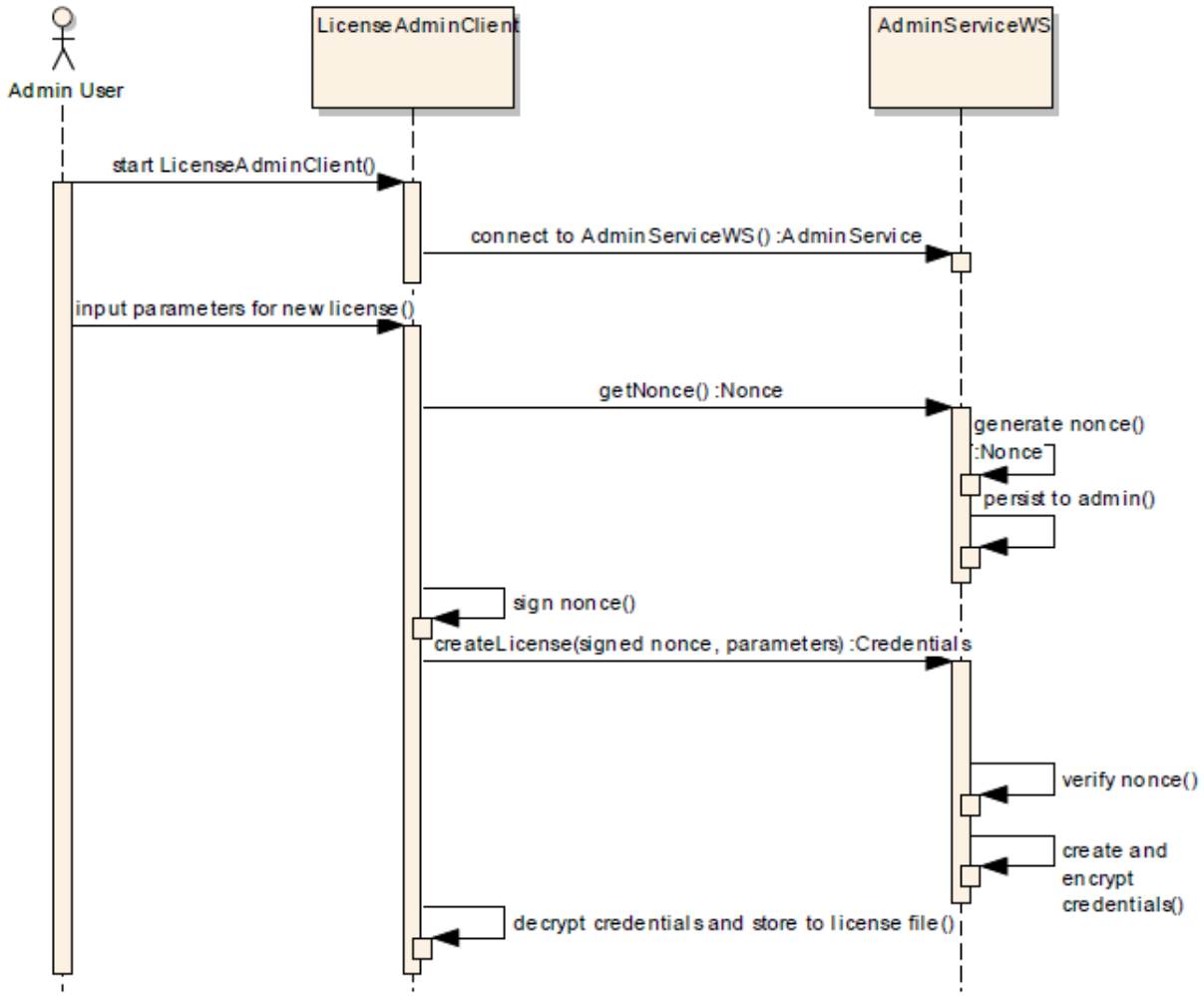


Figure 5: sequence diagram of the create license operation

When using the LicenseAdminClient to retrieve application usage the following operations are performed:

1. Perform a getLicenseesXML operation.

2. Upon success, store the data to a file.

The sequence diagram in Figure 6 illustrates the program flow during this operation.

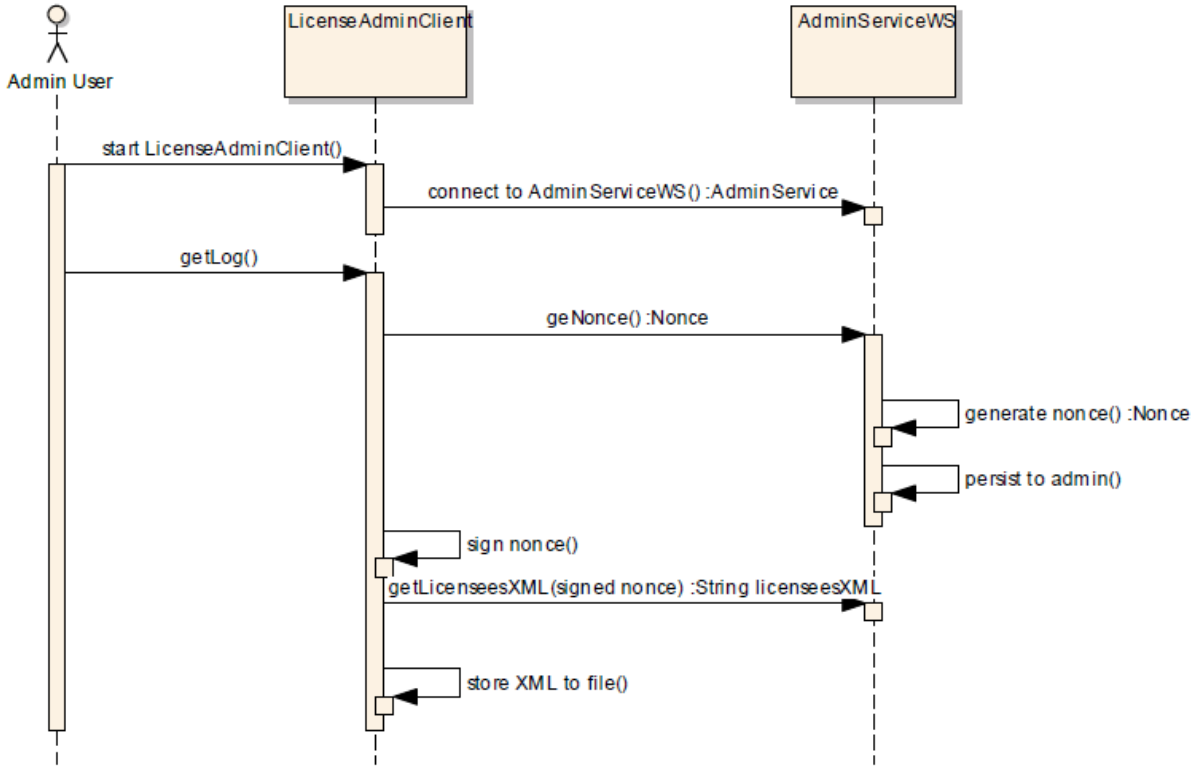


Figure 6: sequence diagram of the get licensees xml operation

3.3 Operation and Maintenance

The request access operation is performed by LicenseManager to LicenseServiceWS by invoking methods on the LicenseService remote object and LicenseManagedApplication object as specified below:

1. Invoke getCredentialsFile() on the LicenseManagedApplication object, and parse the UUID and the private key of the licensee from it.

2. Invoke getNonce() on the LicenseService object, passing on the UUID of the licensee.

3. Create a java.sequrity.SignedObject from the byte array of randomly generated data retrieved from the getNonce() invocation, and sign it with the private key of the licensee.

4. Serialize the signed object and pass it to the LicenseService along with the UUID invoking the requestAccess method, which returns an AccessToken object. If the AccessToken object is null, the verification failed and the LicenseManagedApplication can take appropriate action from the method that is invoked, LicenseManagedApplication.noAccess().

5. From the AccessToken, a String array expressing explicit access to components by their String represented name, can be retrieved by invoking the method getAccess() on the AccessToken object. This array is then passed on to the LicenseManagedApplication implementation by invoking the method setAccess() on the object passing the String array as a parameter.

The create license operation that is performed by LicenseAdminClient is comparable to the request access operation of the LicenseManager although it servers a different purpose:

1. Retrieve the parameters from the admin through command-line. Required parameters are listed below:
    a. The date of license expiration
    b. The name, email and organization of the licensee
    c. A String array with the String representation of the names of the components to which the licensee has explicit access.
2. The name of the license file to which the credentials will be stored upon success.
3. Invoke getNonce on the AdminService object, passing the UUID of the admin licensee.
4. Create a java.sequrity.SignedObject from the byte array of randomly generated data retrieved from the getNonce() invocation, and sign it with the private key of the admin licensee.
5. Serialize the signed object and pass it to the AdminService along with the UUID invoking the createLicense method, which returns a Credentials object. Other parameters that are passed to the AdminService that defines the licensee are;
    a. a Date object, defining the expiration time of the license
    b. a String array defining explicit access to components represented by the String representation of their name
    c. a String that represents the name of the licensee
    d. a String that represents the email of the licensee
    e. a String that represents the organization of the licensee
6. If the Credentials object is null, the operation failed and the AdminService didn’t add a licensee to the database
7. Store the Credentials to a license file that can be shipped with LicenseM anagedApplication software to provide license management.

The getLicenseesXML operation functions as listed below:

1. Invoke getNonce on the AdminService object, passing the UUID of the admin licensee.

2. Create a java.sequrity.SignedObject from the byte array of randomly generated data retrieved from the getNonce() invocation, and sign it with the private key of the admin licensee.

3. Serialize the signed object at pass it to the AdminService along with the UUID invoking the getLicenseesXML method, which returns a String object containing the XML representation of each licensees and their usage as shown below:

```
<xml>
    <Licensee>
        <uuid>String representation of licensee uuid</uuid>
        <country>The country of the licensee</country>
        <email>The email address of the licensee</email>
        <organization>The organization of the licensee</organization>
        <name>The name of the licensee</name>
        <LicenseeLog>
            <ipAddress>The IP address of this licensee log
            entry</ipAddress>
            <loginTime>The time of the request access
            invokation</loginTime>
        </LicenseeLog>
    </Licensee>
<xml>
```

The document can have multiple Licensee tags, one for each licensee, and each Licensee tag can have multiple LicenseeLog tags, one for each request access operation.

3.3.1 Compile, deploy, and test server application

The following sequence of actions gives an explanation from a developer´s point of view on the initial configuration requirements and how to use the deployed server application with the client applications.

Configuring the Application Server

1. Setup a JBoss AS7

2. Setup a PostgreSQL 9.1 database

3. Add PostgreSQL 9.1 database as a DataSource to JBoss AS7

Adding tables to the database

4. Connect to the database in JBoss developer studio

5. Generate tables from entities from the lms-jpa project

6. Export the tables to the database

Compiling the enterprise application

7. Build the lms application to an EAR (Enterprise Archive)

8. Deploy the EAR to the Application Server

Creating a license

9. Run *LicenseAdminClient* in the lms-admin-client project

10. Type *createlicense*

11. Input data

Running an implementation of a license managed application

12. Define the filepath of the retrieved license file in the *getCredentialsFile()* method implemented from the *LicenseManagedApplication* class to the path to the file retrieved in the previous steps.

13. Run *LicenseManagedApplicationImpl* in the *lms-test* project

4 Results

The result of the prototype approach is successful. It has been tested in a test environment on a Windows 7 64-bit PC and provided the proven powerfulness of the JBoss AS7 it delivers a working server application. Testing with the client applications that uses the JAX-WS Runtime are ensuring the capability of the server application. To ensure that the application is error-free further testing is required.

The result is an EAR (Enterprise Archive) that can be deployed to a JBoss AS7 and controlled and monitored via one web service (AdminServiceWS) and used by client nodes (licensees) through another web service (LicenseServiceWS).

The communication between the clients and the web services are protected from a server-side point of view; the server can by cryptography be sure that the clients that connect are the ones that they indicate themselves to be. However, the clients cannot be sure that the server that they connect to is the license server, since it's only identified by a URL. Moreover this can, if necessary, easily be implemented by the use of certificates and possibly SSL connection.

5 Conclusions

Using the JBoss AS7 Java EE implementation provides a stable and straight-forward server designing approach. The ease of using annotations, Enterprise JavaBeans abstracting client connection and entity persistence to the database makes it fun to develop server applications, and the result become powerful since the abstraction of the environment makes it easier to focus on the application flow and usefulness.

Concerning the server application implementation it can be concluded that web services serves a great purpose using SOAP and JAX-WS to interact with client applications, and makes it easy to overview the communication between the components, which gives a powerful application in return when the tools are used in a correct way.

It can also be concluded that the EAR server application can after slight modification be converted to a WAR and deployed to an Apache Tomcat server. Although, Apache Tomcat has not built-in support for JAX-WS, it can be included in the project by adding a few .jar-files and configuring some .xml-files. That is, adding JAX-WS to the web application and configuring it for the endpoint environment.

The fact that the clients cannot verify that the license server that they connect to is the intended one, and might be a fake server, indicates that further security efforts might be to take into consideration. Still, there is no way to protect an uncontrolled host from itself; it can always change whatever might be added to protect it to accomplish undefined behavior.

6 Recommendations for Further Work

To improve the integrity for the license managed application, when connecting to the license server, there may also be some kind of way for the license managed application to be sure that it´s connected to the real server. For this it could have a certificate containing the public key of the license server to be able to verify its genuineness. If this is stripped, like in the above provided theory model, a fake license server can be set up to bypass the need of license. But since the license managed application is run locally, there is no efficient way to protect the host from itself which is why this is stripped. Also, this can be simply implemented in the future.

The license admin client can be further improved by implementing a web interface to control the licenses and functionality can be added to search and monitor the licensees.

The license managed application implementation need further implementation with its initial purpose of usage, the EBITool tool suite, which has according to the length of this thesis, been subordinated from this thesis due to the priority of functionality and proof of concept of the server application. Even so, the prototype has been developed taking this into consideration, making it easily implementable in the tool suite.

7 References

7.1 List of References

Ordered by apperance throughout this document

[1] Oracle Corporation. (2012). *Differences between Java EE and Java SE - Your First Cup: An Introduction to the Java EE Platform*. Available: http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html. Last accessed January 15, 2013.

[2] GlassFish Community. (2013). *The JAX-WS reference implementation in Apache GlassFish*. Available: http://jax-ws.java.net/. Last accessed January 15, 2013.

[3] GlassFish Community. (2013). *GlassFish – Open Source Application Server – Java.net*. Available: http://glassfish.java.net/. Last accessed January 15, 2013.

[4] JBoss community. (2013). *The JBoss community´s JBoss AS 7*. Available: http://www.jboss.org/jbossas/. Last accessed January 15, 2013.

[5] The Apache Software Foundation. (2013). *Apache Tomcat*. Available: http://tomcat.apache.org/. Last accessed January 15, 2013.

[6] The PostgreSQL Global Development Group. (2013). *About PostgreSQL*. Available: http://www.postgresql.org/docs/9.1/static/index.html. Last accessed January 15, 2013.

[7] Redhat Inc. (2013). *JBoss Developer Studio web site*. Available: https://devstudio.jboss.com/download/. Last accessed January 15, 2013.

[8] Microsoft Corporation. (2013). *Windows 7 web site*. Available: http://windows.microsoft.com/en-US/windows7/products/home. Last accessed January 15, 2013.

[9] Bombardier Inc. (2011). *About Bombardier Transportation*. Available: http://www.bombardier.com/en/transportation/about-transportation. Last accessed January 15, 2013.

[10] Oracle Corporation. (2011). *The Java API*. Available: http://docs.oracle.com/javase/6/docs/api/. Last accessed January 15, 2013.

[11] Sam Ruby. (2009). *Sam Ruby Blogging on Cryptographic Nonce with an implementation* Available: http://www.intertwingly.net/blog/1585.html. Last accessed January 15, 2013.

[12] Bruce Schneier. (2004). *Crypto-Gram Newsletter – Security Notes from All Over: Man-in-the-Middle Attack*. Available: http://www.schneier.com/crypto-gram-0404.html#6. Last accessed January 15, 2013.

[13] Schlichtherle IT Services. (2012). *True License - open source license management for closed source Java applications*. Available: http://truelicense.java.net/. Last accessed January 15, 2013.

[14] Oracle Corporation. (2013). *Enterprise JavaBeans Technology* Available: http://www.oracle.com/technetwork/java/javaee/ejb/index.htmlhttp://truelicense.java.net/. Last accessed January 15, 2013.

[15] W3C Community. (2007). *SOAP as specified by the W3C* Available: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. Last accessed January 15, 2013.

[16] Oracle Corporation. (2013). *UUID as specified in the Java API*. Available: http://docs.oracle.com/javase/6/docs/api/java/util/UUID.html. Last accessed January 15, 2013.

[17] Oracle Corporation. (2013). *String representation of the UUID as specified in the Java API*. Available: http://docs.oracle.com/javase/6/docs/api/java/util/UUID.html#toString(). Last accessed January 15, 2013.

[18] EMC Corporation. (2012). *Public-Key Cryptography Standards as specified by RSA Laboratories, founded at the company of the inventors of RSA public-key cryptosystem*. Available: http://www.rsa.com/rsalabs/node.asp?id=2124. Last accessed January 15, 2013.

[19] Oracle Corporation. (2010). *Java API reference to the class used for secure one-way hash functions, such as SHA-1*. Available: http://docs.oracle.com/javase/1.5.0/docs/api/java/security/MessageDigest.html. Last accessed January 15, 2013.

[20] JBoss Community. (2013). *Hibernate – Relational Persistence for Java and .NET*. Available: http://www.hibernate.org. Last accessed January 15, 2013.

[21] Oracle Corporation. (2013). *Java Persistence API web site*. Available: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html. Last accessed January 15, 2013.

[22] Oracle Corporation. (2013). *Java SE web site*. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html. Last accessed January 15, 2013.

7.2 Personal Contacts

Alphabetically ordered by surname

- Christian Andersson – has, as examiner, enlightened about the requirements and goals of this thesis, as well as informative comments about improvements.
- Anders Ask, HKr – has provided necessary templates, documents throughout the course in order to fulfill the requirements of this document.
- Henrik Jönsson, Bombardier Transportation – has, as instructor, provided the necessary information concerning the approach, aim and purpose of this thesis.
- Andreas Nilsson, HKr – has, as instructor, provided guidance, comments and views towards refining, improving and finishing this thesis.
- Fredrik Salomonsson, Bombardier Transportation – has through consultation in meetings enlightened on certain aspects and questions throughout the thesis with his expertise on the subject.

7.3 Bibliography

Alphabetically ordered by surname

- Daigneau, R. (2011). *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. Addison-Wesley Professional, ISBN 0321544209
- Elliott, J., Fowler, R., & O'Brien T. (2008). *Harnessing Hibernate*. O'Reilly Media, ISBN 0596517726
- Gupta, A. (2012). *Java EE 6 Pocket Guide*. O'Reilly Media, ISBN 1449336684
- Hsu, L., & Hsu, L. (2012). *PostgreSQL: Up and Running*. O'Reilly Media, ISBN 1449326331
- Keith, M., Schincariol, M., & DeMichiel, L. (2009). *Pro JPA 2: Mastering the Java Persistence API*. APRESS, ISBN 1430219564
- Lee Rubinger, A., Burke, B., & Monson-Haefel, R. (2010). *Enterprise JavaBeans 3.1*. O'Reilly Media, ISBN 0596158025
- Schneier, B. (1996). *Applied Cryptography Second Edition: protocols, algorithms, and source code in C*. John Wiley & Sons, ISBN 0-471-12845-7.

Appendix A Project Overview

Overview and description of projects, packages and classes

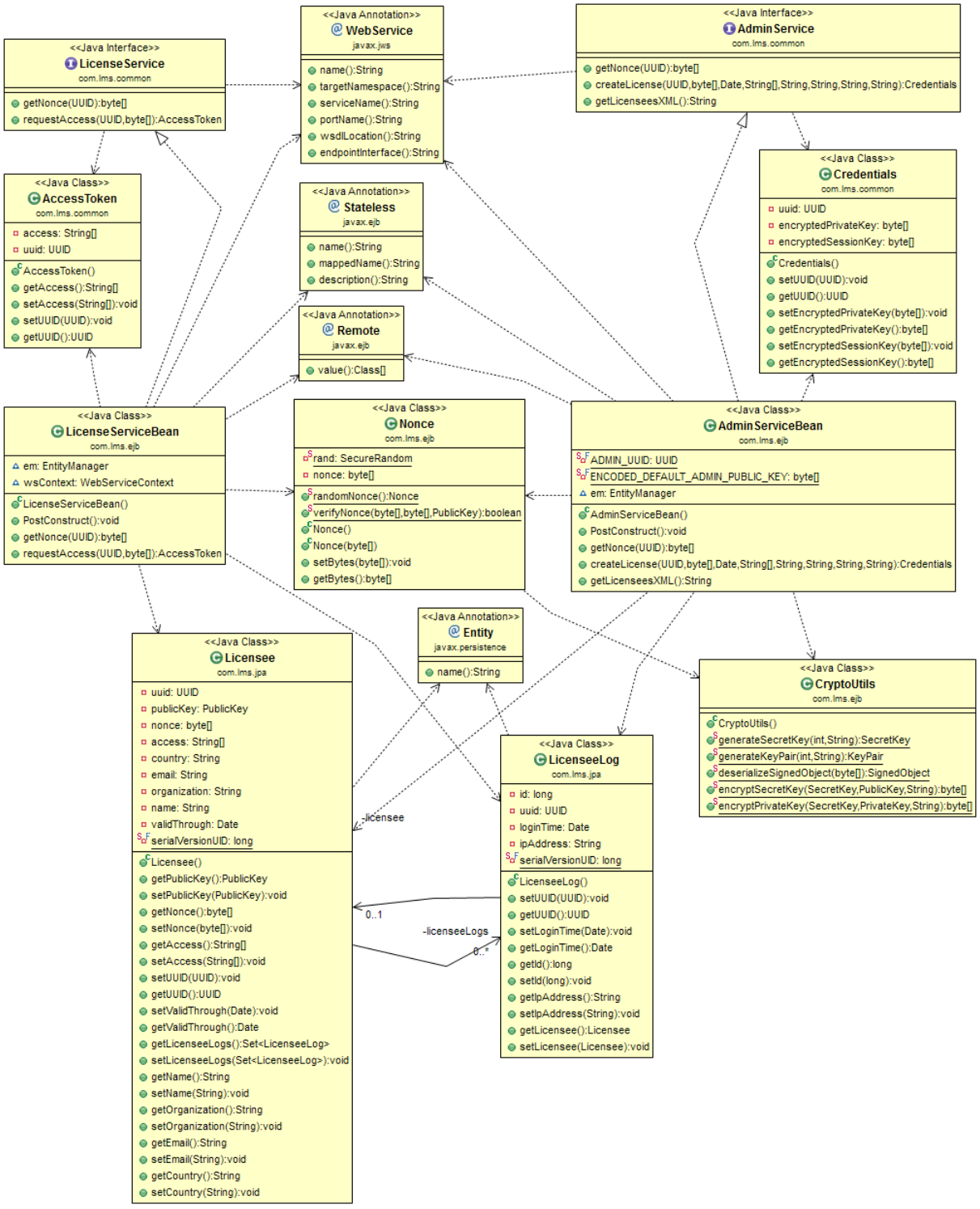
Project	Description	Packages	Requires	Classes	Class Description	Location
lms	The umbrella project of the license server application	none	lms-ejb lms-ejbClient lms-jpa	none		server-side
lms-ejb	The business logic of the web services	com.lms.ejb	none	AdminServiceBean	The business logic of the AdminService WS web service	server-side
				CryptoUtils	Static methods coherent with cryptograhy, and serialization	
				LicenseServiceBean	The business logic of the LicenseServiceWS web service	
				Nonce	A wrapper class for the Nonce, including static methods of generating and verifying nonce	
lms-ejbClient	The client project of the server application, that also needs to be on the client side	com.lms.common	none	AccessToken	Wrapper class for the response on a request access operation on the LicenseServiceWS	server-side and client-side
				AdminService	The interface representing the AdminService WS web service	
				Credentials	Wrapper class for the credentials that are sent to the LicenseAdminClient upon creating a license	

				LicenseService	The interface representing the LicenseServiceWS web service	
lms-jpa	The JPA project, containing entities that are mapped to the DataSource	com.lms.jpa	none	Licensee	The licensee entity, representing a single licensee	server-side
				LicenseeLog	The licensee log entity, representing a single access request of a license	
lma	The project of which the .jar-file is included to the project that is to be license managed	com.lms	lms-ejbClient lms-utils	LicenseManagedApplication	The interface that is implemented by the license managed application	client-side
				LicenseManager	The thread that handles the connection to the LicenseServiceWS web service	
lms-utils	Utils used lma and lma-admin-client	com.lms.utils	none	CryptoUtils	Utility class with static methods to decrypt data from and serialize objects to be passed to the server application	client-side
lms-test	Implementation of a license managed application that uses a license file and connects to LicenseServiceWS	default package	lma	LicenseManagedAppImpl	This class implements the LicenseManagedApplication interface and starts the LicenseManager thread providing license management to itself	client-side
lms-admin-client	An easy CLI implementation of the license admin client, used to create licenses, and view licensee log	default package	lms-ejbClient lms-utils	LicenseAdminClient	Implements the logic of an admin client, taking parameters from the command line, creates license, and views log	client-side

Appendix B Class Diagram

Class Diagram of License Management Server

This diagram shows the classes of the license management server, the enterprise server application that is running on the JBoss Application Server. It shows their properties and functions and their inheritances and dependencies to each other. Some Java EE annotations are also illustrated to explain which is what.



Appendix C Component Diagram

Component Diagram of License Management System

This diagram explains how all components of the system are interacting with each other. In the prototype, the EBITool component is replaced by a stub that is acting as a license managed application.