Bluenog Platform BluenogCMS Administrator's Guide

Document Title:	Bluenog Platform BluenogCMS Administrator's Guide
Document Version:	3.1.9sp1
Effective Date:	March 31 2008

Revision Details

Version	Date	Prepared By:	Description
2.1.2	7/27/2007	Adam Steidley	Initial version

1. Introduction to BluenogCMS

Enterprise Content Management (ECM) systems are responsible for helping an organization to better create, edit, collect, manage and distribute their content. Let us first examine how BluenogCMS approaches these tasks.

BluenogCMS stores and maintains content as XML documents. These documents are authored by business users with web based forms. The documents are then placed into a workflow where the documents are approved for publication. Finally the documents are available for access for other channels, including portals, websites, data feeds, and external systems.

There are three principal applications that will need to be deployed to support these operations, the Content Repository, the CMS User Interface and the Content Access Methods.

1.1. The Content Repository

The Content Repository physically stores the XML documents and allows for access to them. Fundamentally the Bluenog Content Repository is a WebDAV server. WebDAV, an extension to the HTTP protocol, adds properties to each document as well as the ability to perform searches. From a functional perspective, communication with the repository is handled as HTTP. The documents and their associated metadata can be stored either on the file system or in a database. File system storage is recommended only for development. Production installations should use either a MySQL or Oracle database for storage. Searches are processed by a Lucene search engine. It will create an index for the content to speed the performance of searches. The repository relies on JMS to publish events corresponding to changes in the content stored. External systems and APIs can use this mechanism to update their caches in near real time. Security for the User Interface and the Content Access Methods are handled through the repository. The repository can store the user information itself or it can be connected to an LDAP repository. The repository can be run either as a deployed WAR or as a stand alone Java application.

1.2. The CMS User Interface

Users will not interact directly with the Content Repository when they need to maintain the content. The CMS User Interface facilitates the user interaction with the content. It will also be run either as a deployed WAR or a standalone application and the user interaction is through a web browser as with any web application. The CMS UI presents a view of the content hierarchy as simple folder tree. When a piece of content is edited, a custom form is presented. This allows for the use of text boxes, drop down lists and pop-up pickers as needed. The content is validated against both its schema (xsd) and any additional rules that are provided. If successfully validated, the resultant XML is stored into the repository. The CMS UI also subscribes to the repository's JMS queue to keep its internal data synchronized. The CMS User Interface also handles all of the workflow functions. These workflow functions are maintained with OSWorkflow, an open source workflow project. It requires a connection to the database to store the current state

information for the workflows. There are a number of internal schedulers, each of which will need access to the database.

1.3. Content Access Methods

There are a number of methods to access the content from the repository. The Client Adapter Java API is the primary mechanism for doing so. It can be used directly or through the Content SPI for WebLogic Portal or through the supplied J2EE Tag Library. The Adapter will retrieve its content from the repository over HTTP with a combination of HTTP GET's, WebDAV PROPFIND's and WebDAV searches. These results can be explicitly retrieved from the repository at each request or the results can be stored in a cache on the client side. The cache used is from the JCS project under Jakarta. It supports a least recently used (LRU) algorithm and can be configured to expire items by both item count and memory used. This cache can also be configured to work with the repository's JMS topic to update itself when the content is changed.

2. Custom code

Code that is custom to a customer installation is kept out of the *bluenog-repository.war* and out of the *bluenog-cms.war*. To do this, an external directory is created and the configuration files are kept in these directories. As a standard the environment variable BLUENOG_CLASSPATH is set to the root configuration directory. Now the environment variable is added to the CLASSPATH on the server startup scripts.

The two properties files that are referenced for all Bluenog modules are *bluenog.properties* and *local.bluenog.properties*. The values from bluenog.properties are intended to be those that will be valid in all of your installations. Any values specific to the server (or environment) should be placed in the *local.bluenog.properties* file. The values from local will be loaded on top of the values from the *bluenog.properties* file.

All of the applications will need access to the content type definitions. In the *bluenog.properties* file the following variables need to have the {base types dir} portion replaced with the directory of the *types.xml* file. If Cocoon extensions are used in the CMS, then the {customer sitemap} portion needs to be replaced with the directory where these files reside.

```
# Location of CMS content type definitions
cms.types.dir=file:{base types dir}
cms.types.dir.plain={base types dir}
bluenog.customerSitemap=file:{customer sitemap}
```

3. Configuring the Repository

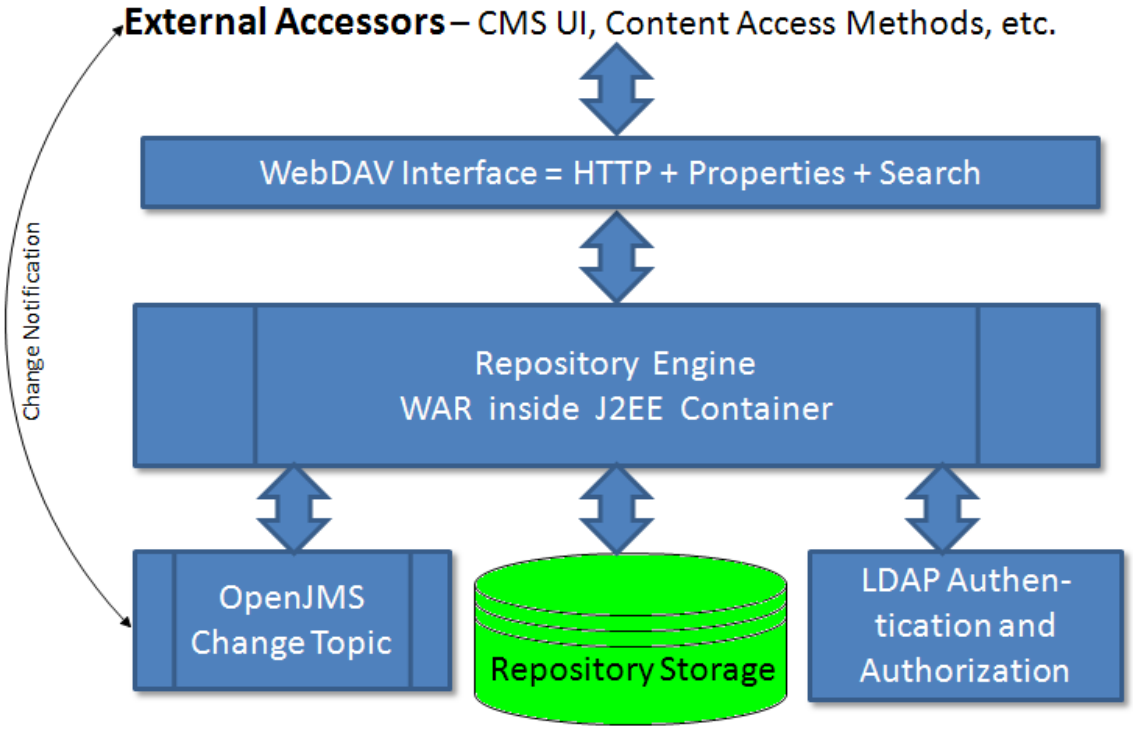

Figure 1 Repository configuration

3.1. Overview

The repository will need three infrastructure pieces configured for it.

1. **The database -** To allow for storage of the content data, will be accessed through a connection pool made available through JNDI.
2. **The JMS implementation** - OpenJMS is included in the WAR itself to supply the JMS Topics for content change notification. Access to this information will again be through JNDI.
3. **LDAP** - Can be used as the data store for user authentication and authorization. If this is not required, the repository can store the data for itself.

3.2. Install the War to the Server

The first step is to install the *bluenog-repository.war* to the server. It is recommended to install the repository to the server context of /default.

3.2.1. Tomcat

Add jars to common/endorsed:

- Xalan-2.7.0.jar
- XercesImpl-2.7.1.jar
- Xml-apis-1.3.02.jar

Add jars to common/lib

- mysql-connector-3.0.16.jar

Add BLUENOG_CLASSPATH to the CLASSPATH. This is in bin/setclasspath.sh (or .bat on Windows)

3.2.2. Weblogic Server

Specify <JDK_HOME> when creating the domain

Add jars to <JDK_HOME>lib/endorsed:

- xalan-2.6.0.jar
- xerces-2.4.0.jar
- xmlParserAPIs-2.0.2.jar

If you are using Weblogic Server 8.1sp6, you will need to apply patch CR287255_810sp6.jar. This is typically done by adding this jar to the front of the CLASSPATH in the setDomainEnv.sh (or setDomainEnv.cmd on Windows). Also, you must add the attribute *EnforceValidBasicAuthCredentials* in the <SecurityConfiguration> block and set it equal to false.

```
<SecurityConfiguration

        CredentialEncrypted="{3DES}f4QcrDX6l+d1R0E68lQly"

        EnforceValidBasicAuthCredentials="false" Name="repoDomain"
RealmBootStrapVersion="1"/>
```

Now add BLUENOG_CLASSPATH (the directory containing your bluenog.properties files) to the server classpath. This will allow for the bluenog.properties files to be found at runtime.

Note: the endorsed folder needs to be created before copying the 3 jar files

3.3. Create the Database

The slide database will need to be created in your database. Executing the following sql scripts will create these with user and database names that align with the default JNDI properties.

3.3.1. MySQL

```
mysql --user root < mysqlReposUser.sql
mysql --user slide --password  slide < MySqlSchema.sql
```

3.3.2. Oracle

```
sqlplus sys/password as SYSDBA @bluenog_repos_oracle.sql
sqlplus slide/secret OracleSchema.sql
```

3.4. Configure JNDI - Database Connection

3.4.1. Tomcat

The JNDI data source jdbc/repositoryDataSource will need to be defined. In tomcat this can be done by adding the following to apache-tomcat-5.5.20/conf/Catalina/localhost/default.xml

```
<Context path="/default" docBASE="${catalina.home}/webapps/default"
antiJARLocking="true">
    <Resource
    name="jdbc/repositoryDataSource"
    type="javax.sql.DataSource"
    driverClassName="com.mysql.jdbc.Driver"
    maxIdle="2"
    maxWait="5000"
    username="slide"
    password="secret"
    url="jdbc:mysql://localhost:3306/slide?autoReconnect=true"
    maxActive="12"/>
</Context>
```

3.4.2. Weblogic Server

Sample of the JDBC connections to Oracle from config.xml:

```
<JDBCConnectionPool DriverName="oracle.jdbc.driver.OracleDriver"
Name="BluenogRepoConnectionPool" PasswordEncrypted="{3DES}SkSkbHwgbEE="
Properties="user=slide" Targets="ms1,ms2" TestTableName="SQL SELECT 1 FROM
DUAL" URL="jdbc:oracle:thin:@192.168.1.25:1521:ORADEV"/>

<JDBCDataSource JNDIName="jdbc/repositoryDataSource"
Name="jdbc/repositoryDataSource" PoolName="BluenogRepoConnectionPool"
Targets="portalCluster"/>
```

Create a connection pool using the Weblogic supplied oracle driver. Also create a Data Source using this connection pool. The WAR is expecting the data source JNDI Name to be:

- jdbc/repositoryDataSource

3.5. Configure JMS

The Cocoon engine uses a JMS subscription to be notified of Repository changes. In some cases, Cocoon can lose the JMS connection with the Repository, for example if the Repository is restarted or the network is down. The Cocoon cache will then fail to get any subsequent JMS messages. To mitigate this, an auto-reconnect behavior has been turned on.

3.5.1. Tomcat

The Tomcat distribution uses an internal jms provider that is preconfigured when you install the WAR. No additional configuration needs to be performed.

3.5.2. Weblogic Server

Create JMS Connection Factory. The WAR is expecting the JNDI name to be:

- jms/repositoryTCF

Now create a topic. In the following sample code, this was done as a distributed topic on a Weblogic cluster. The WAR expects the JNDI name to be:

- jms/repositoryTopic

Below shows the values from the config.xml:

```
<JMSConnectionFactory JNDIName="jms/repositoryTCF" Name="jms/repositoryTCF"
Targets="portalCluster"/>

<JMSDistributedTopic JNDIName="jms/repositoryTopic" Name="jms/repositoryTopic"
Targets="portalCluster" Template="jms/repositoryTopic">

<JMSDistributedTopicMember JMSTopic="jms/repositoryTopic@cgJMSServer_auto_1"
Name="jms/repositoryTopic@cgJMSServer_auto_1"/>

<JMSDistributedTopicMember JMSTopic="jms/repositoryTopic@cgJMSServer_auto_2"
Name="jms/repositoryTopic@cgJMSServer_auto_2"/>

</JMSDistributedTopic>
```

3.6. Configure LDAP (optional)

LDAP offers an industry standard way to authenticate and authorize access to the repository. If your organization already has and LDAP directory, you can leverage that infrastructure by mapping LDAP elements to existing roles in the repository.

First you'll need to enable the use of the LDAP configuration in *components.xconf*, found in the WEB-INF directory of your WAR deployment.

```
...
<webdav activation="inline" id="webdav" logger="slide.webdav">
    <jndi use="true">config/ldap-realm.xml</jndi>
...
```

The location of the primary LDAP configuration file is specified here. Here is an example of that file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is an example configuration file -->
<realms>
  <namespace name="default">
    <!-- cache time in milliseconds -->
    <cache-time>5000</cache-time>
    <!-- connection setup, super-user needs complete distinguished
name! -->
    <super-user>cn=ldapadmin,dc=bluenog,dc=com</super-user>
    <super-user-password>password</super-user-password>
    <!-- Location of LDAP server and port -->
    <provider-url>ldap://10.0.0.10:389</provider-url>
```

```
    <!-- Authentication type i.e. simple, MD5, etc -->
    <authentication>simple</authentication>

    <!-- user discovery -->
    <username-attribute>cn</username-attribute>
    <password-attribute>userPassword</password-attribute>
    <distinguished-name-attribute>dn</distinguished-name-attribute>
    <user-search-root>ou=devdept,dc=bluenog,dc=com</user-search-root>

    <method>bind</method>
  </namespace>
</realms>
```

This file establishes the basic requirements to connect to and browse the LDAP directory for users of the system. You can utilize LDAP filters to establish a particular section of your directory to search for users, as specified by RFC 2254. The filter block must be placed withing the <namespace> tag. Here is an example of the syntax within the XML configuration file:

```
<filters>
    <filter>
          <![CDATA[(&(objectClass=inetOrgPerson)(cn=*))]]>
      </filter>
</filters>
```

Once this is configured you need to establish two JNDI stores in the definition.xml, found in the slide directory underneath WEB-INF in your WAR deployment. Here are the sections you need to add. Please note the environment specific parameters that you will need to change to fit your infrastructure.

```
<store name="users">
          <nodestore
classname="org.apache.slide.store.txjndi.JNDIUsersStore">

          <parameter name="cache.refresh.checkrate">15</parameter>
          <parameter name="cache.refresh.rate">600</parameter>
          <parameter name="cache.refresh.threshold">500</parameter>
          <parameter
name="jndi.container">OU=employees,DC=bluenog,DC=com</parameter>
          <parameter name="jndi.attributes.user">cn</parameter>
          <parameter name="configuration">ldap-users.xml</parameter>
          <parameter name="java.naming.provider.url">ldap://<ldap-
server>:389</parameter>
          <parameter
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</
parameter>
```

```
          <parameter
name="java.naming.security.principal">cn=ldapadmin,dc=bluenog,dc=com<
/parameter>
          <parameter
name="java.naming.security.authentication">simple</parameter>
          <parameter
name="java.naming.security.credentials">password</parameter>

          </nodestore>

          <securitystore
classname="org.apache.slide.store.txfile.TxXMLFileDescriptorsStore">
          <parameter name="rootpath">slide/metadata</parameter>
          <parameter name="workpath">slide/metadata</parameter>
          </securitystore>
          <lockstore>
          <reference store="securitystore"/>
          </lockstore>
          <revisiondescriptorsstore>
          <reference store="nodestore"/>
          </revisiondescriptorsstore>
          <revisiondescriptorstore>
          <reference store="nodestore"/>
          </revisiondescriptorstore>
          <contentstore>
          <reference store="nodestore"/>
          </contentstore>
</store>
<scope match="/users" store="users"/>

<store name="groups">
          <nodestore
classname="org.apache.slide.store.txjndi.JNDIRolesStore">
          <parameter name="cache.refresh.checkrate">15</parameter>
          <parameter name="cache.refresh.rate">600</parameter>
          <parameter name="cache.refresh.threshold">500</parameter>
          <parameter name="jndi.attributes.user">cn</parameter>
          <parameter name="configuration">ldap-roles.xml</parameter>
          <parameter name="java.naming.provider.url">ldap://<ldap-
server>:389</parameter>
          <parameter
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</
parameter>
          <parameter
name="java.naming.security.principal">cn=ldapadmin,dc=bluenog,dc=com<
/parameter>
```

```
          <parameter
name="java.naming.security.authentication">simple</parameter>
          <parameter
name="java.naming.security.credentials">password</parameter>
          </nodestore>
          <securitystore
classname="org.apache.slide.store.txfile.TxXMLFileDescriptorsStore">
          <parameter name="rootpath">slide/metadata</parameter>
          <parameter name="workpath">slide/metadata</parameter>
          </securitystore>
          <lockstore>
          <reference store="securitystore"/>
          </lockstore>
          <revisiondescriptorsstore>
          <reference store="nodestore"/>
          </revisiondescriptorsstore>
          <revisiondescriptorstore>
          <reference store="nodestore"/>
          </revisiondescriptorstore>
          <contentstore>
          <reference store="nodestore"/>
          </contentstore>
</store>
<scope match="/roles" store="groups"/>
```

Now in the same location as the *ldap-realm.xml* file, you'll want to edit *ldap-roles.xml* and *ldap-users.xml* to map LDAP members to the repository users list and to repository roles. In the ldap-roles configuration the group name must be the same as an existing role in the repository. You can establish these roles via the Dashboard in CMS.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is an example ldap-roles.xml configuration file -->
<groups>
  <group name="root">
    <member>/users/admin</member>
    <member>/users/root</member>
  </group>
  <group name="user">
    <search dn="ou=employees,dc=bluenog,dc=com">
    </search>
  </group>

  <group name="administrators">
    <member>/users/admin</member>
  </group>
  <group name="SalesEditors">
```

```
    <search dn="ou=employees,dc=bluenog,dc=com">

<filter><![CDATA[(&(departmentNumber=SAL01)(!(bluenogManager=*)))]]><
/filter>
    </search>
  </group>
  <group name="TechEditors">
    <search dn="ou=employees,dc=bluenog,dc=com">

<filter><![CDATA[(&(departmentNumber=PRD01)(!(bluenogManager=*)))]]><
/filter>
    </search>
  </group>
  <group name="SalesAuthors">
    <search dn="ou=employees,dc=bluenog,dc=com">

<filter><![CDATA[(&(departmentNumber=SAL01)(bluenogManager=*))]]></fi
lter>
    </search>
  </group>
  <group name="TechAuthors">
    <search dn="ou=employees,dc=bluenog,dc=com">

<filter><![CDATA[(&(departmentNumber=PRD01)(bluenogManager=*))]]></fi
lter>
    </search>
  </group>
</groups>
```

Here is a sample *ldap-users.xml* configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is an example ldap-users.xml configuration file -->
<users>
  <search dn="ou=employees,dc=bluenog,dc=com">
  </search>
  <user name="root" pass="password"/>
  <user name="admin" pass="password"/>
  <user name="siteuser" pass="siteuser"/>
</users>
```

You can employ LDAP Filters in the search block. Please note that individual filters will be ORed together, however within the filter, you can utilize standard LDAP filter syntax to create whatever subsets you may need.

3.7. Initialize Repository

The repository will need to be initialized. Once the repository is running, you will need to execute the following. Check sample.properties to guarantee that the ip, port, username and password used for your repository is set correctly.

```
java -jar repository-preparation-uber.jar sample.properties
```

3.8. Clustering (optional)

Clustering of the repository allows you to distribute the load of requests to the repository from your production site. Typically this is configured as a cluster of two or more nodes pointing to the same RDBMS back-end.

You first need to enable the configuration in the *components.xconf* file, found in the WEB-INF directory of your WAR deployment, as follows:

```
<jgroups-cluster-propagator id="jgroups-cluster-propagator"
logger="slide.jgroups-cluster">
    <enabled>true</enabled>
          <stack src="<absolute-location-to-config-file>"/>
          <group name="<unique-cluster-name>"/>
</jgroups-cluster-propagator>
```

Then you will need to establish your environment specific parameters in the XML config file that you specified above. Here is an example config file:

```
<?xml version="1.0"?>
<config>
     <TCP
          bind_addr="<node-ip>"
          start_port="5055"/>
     <TCPPING
          initial_hosts="<node1-ip>[5055],<node2-ip>[5055],..."
          port_range="5"
          timeout="3000"
          num_initial_members="3"
          up_thread="true"
          down_thread="true"/>
     <MERGE2
          min_interval="5000"
          max_interval="10000"/>
     <FD_SOCK/>
     <VERIFY_SUSPECT
          up_thread="false"
          down_thread="false"
          timeout="1500"/>
     <pbcast.NAKACK
          up_thread="false"
```

```
            down_thread="false"
            max_xmit_size="8096"
            gc_lag="50"
            retransmit_timeout="600,1200,2400,4800"/>
        <UNICAST
            timeout="600,1200,2400,4800" />
        <pbcast.STABLE
            up_thread="false"
            down_thread="false"
            desired_avg_gossip="20000"/>
        <FRAG
            frag_size="8096"
            down_thread="false"
            up_thread="false"/>
        <pbcast.GMS
            up_thread="false"
            down_thread="false"
            join_timeout="5000"
            join_retry_timeout="2000"
            shun="false"
            print_local_addr="true"/>
</config>
```

3.9. Replication (optional)

You can configure your repository to replicate its data to another repository. This configuration can be on all or a portion of the data in the repository. The repository will typically hold the published content as well as the content that is work in process. It may be useful to replicate the live content to a repository in an offsite location. This is accomplished by editing the replicators.xml file in the WEB-INF/slide directory. Entries like the below will cause all writes to the repository to be mirrored to the remote repository.

```
<replicators>
  <replicator uri="/path/to/replicate">
    <location>
      <host>localhost</host>
      <port>60000</port>
      <user>username</user>
      <password>password</password>
      <realm>auth realm</realm>
      <rootpath>/rootpath</rootpath>
      <path>/target/path</path>
    </location>
  </replicator>
</replicators>
```

3.10. Bluenog properties

The following values need to be set correctly in your bluenog.properties file. This will tell the repository where the content type definitions are. Replace the {base types dir} with the location of the content types definition file types.xml. Remember that the values in local.bluenog.properties take precedence over the values in bluenog.properties.

```
# Location of CMS content type definitions
cms.types.dir=file:{base types dir}
cms.types.dir.plain={base types dir}
```

4. Configuring the CMS User Interface

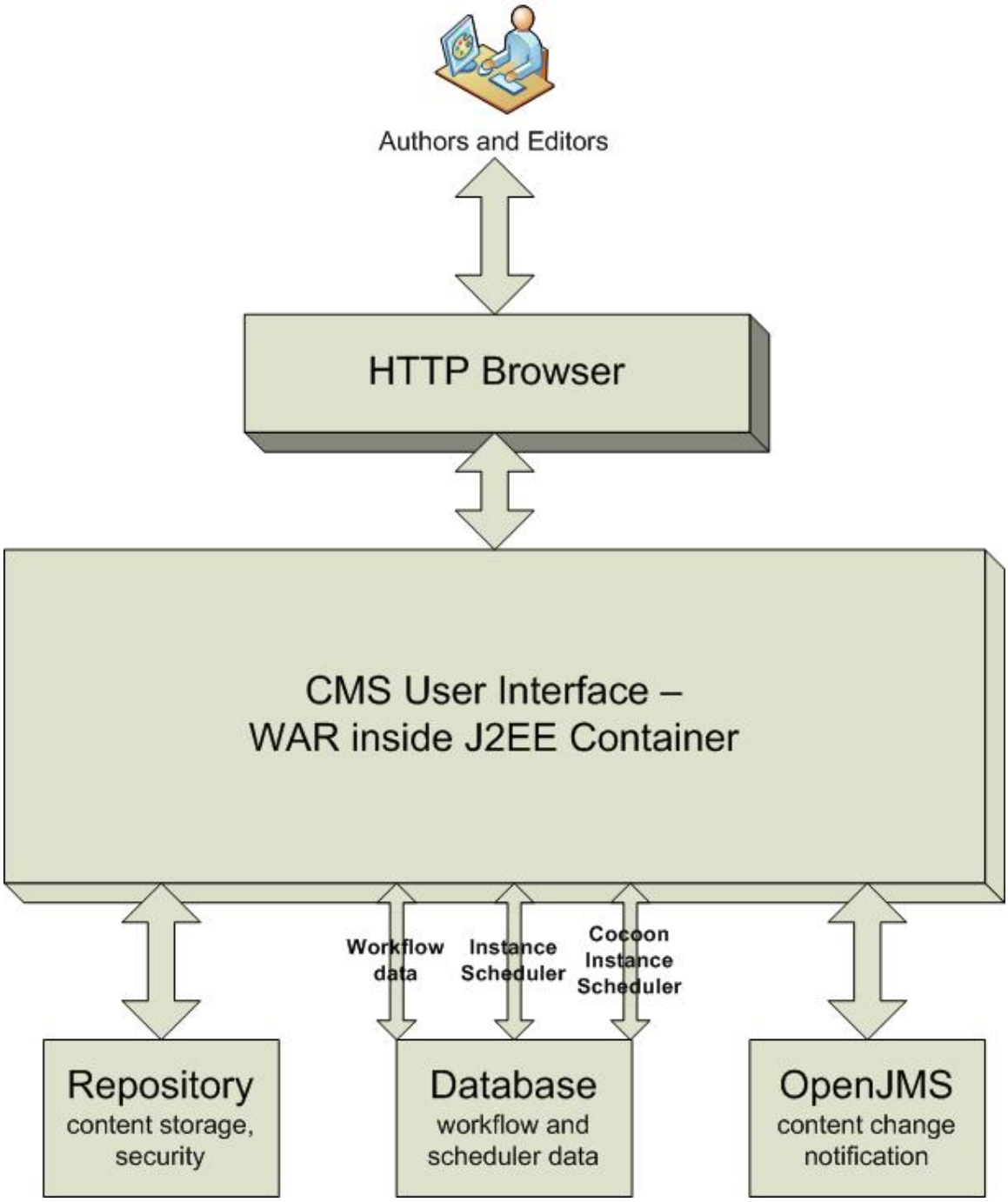

Figure 2 CMS User Interface configuration

4.1. Overview

The CMS User Interface will run as a Web Application and be where the users log in to the system to maintain the content. It needs access to the Repository, both over WebDAV and JMS. It will require connections to the database for project data around workflow and scheduling and for 2 internal schedulers.

4.2. Install the War to the Server

The first step is to install the bluenog-cms.WAR to the server. The WAR must be deployed to the ROOT context ("/").

4.2.1. Tomcat

Add jars to common/endorsed:

- Xalan-2.7.0.jar
- XercesImpl-2.7.1.jar
- Xml-apis-1.3.02.jar
- bcel-5.1.jar
- geronimo-spec-jms-1.0-M1.jar
- jakarta-regexp-1.4.jar

Add jars to common/lib

- mysql-connector-3.0.16.jar

Add BLUENOG_CLASSPATH to the CLASSPATH. This is in bin/setclasspath.sh (or .bat on Windows)

4.2.2. Weblogic

Add jars to the preclasspath:

- XercesImpl-2.9.0.jar
- Xml-apis-1.3.04.jar
- bcel-5.1.jar
- jakarta-regexp-1.4.jar

Add BLUENOG_CLASSPATH to the CLASSPATH. This is in setDomainEnv.sh (or .cmd on Windows)

4.3. Create the Databases

The CMS needs database connections for 1 scheduler and for workflow related data. The database for one scheduler and for the workflow data will be shared by all instances of the CMS that run as a logical group. This data is put into the projectdata schema by the default scripts. The bluenog_cms_mysql script will create the username's and schemas that we used by the scripts. There are equivalent scripts to be used for Oracle.

```
mysql --user root < bluenog_cms_mysql.sql
mysql --user root  -D projectdata < quartz.sql
mysql --user root  -D projectdata < osworkflow.sql
```

4.4. Repository Access

The locations of the repository is read from the bluenog.properties files. If all of your CMS servers are using the same Repository, then this information should be placed in your bluenog.properties. If it is unique for each server, then just place the items that will differ in the local.bluenog.properties. The username and password used to login to the

repository can also be changed. The repository name is the context that the repository was deployed to.

```
# ---- Repository location used by the CMS ---- #
maven.cocoon.repository.name=default
maven.cocoon.repository.port=8080
maven.cocoon.repository.host=localhost
maven.cocoon.repository.authenticationrealm=default realm
maven.cocoon.repository.systemcredentials.username=root
maven.cocoon.repository.systemcredentials.password=password
```

4.5. Configure JNDI

4.5.1. Database Connections

The JNDI data source jdbc/siteProjectData will need to be defined. In Tomcat this can be done by adding the following to apache-tomcat-5.5.20/conf/Catalina/localhost/ROOT.xml

```
<Context path="" docBASE="${catalina.home}/webapps/ROOT"
antiJARLocking="true">
    <Resource
    name="jdbc/siteProjectData"
    type="javax.sql.DataSource"
    driverClassName="com.mysql.jdbc.Driver"
    maxIdle="2"
    maxWait="5000"
    username="bluenoguser"
    password="bluenogpass"
    url="jdbc:mysql://localhost:3306/projectdata?autoReconnect=true"
    maxActive="10"/>
</Context>
```

4.5.2. JMS Connections

The following jms settings are required in the bluenog.properties file for the CMS to obtain JMS events to update its cache properly. This assumes that the repository is being run under the default settings.

```
jms.topic=jms/repositoryTopic
jms.connection=local
jms.listener=local
jms.connection-factory=TCPTopicConnectionFactory
jms.jndi.initial-factory=org.exolab.jms.jndi.InitialContextFactory
jms.jndi.provider=tcp://localhost:3035/
jms.subscription=cms_local
```

4.5.3. JMS Connections under Weblogic

If you have your repository deployed to the same Weblogic environment as your CMS, then your JMS connections are already setup. The expected bluenog.properties settings are:

```
#----- JMS configuration used by the CMS with Weblogic -----#
jms.topic=BluenogRepo/repositoryTopic
jms.connection-factory=repositoryTCF
jms.jndi.initial-factory=weblogic.jndi.WLInitialContextFactory
jms.jndi.provider=t3://localhost:7001/
```

4.6. Bluenog properties

The following values need to be set correctly in your bluenog.properties file. This will tell the CMS where the content type definitions are. Replace the {base types dir} with the location of the content types definition file types.xml. The customerSitemap setting is the location of any cocoon extensions that are created.

```
# Location of CMS content type definitions
cms.types.dir=file:{base types dir}
cms.types.dir.plain={base types dir}
bluenog.customerSitemap=file:{customer sitemap}
```

5. Configuring the Content Access

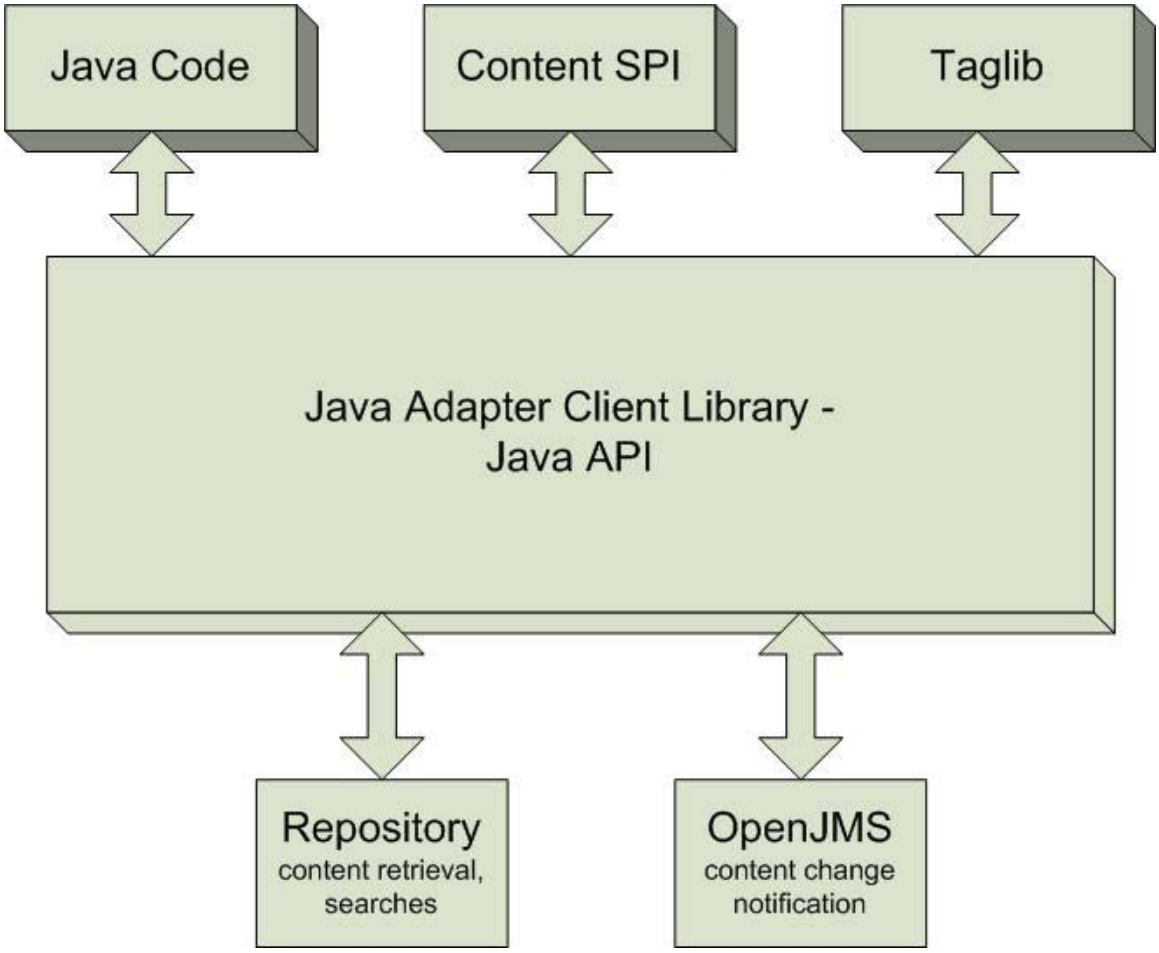

Figure 3 Content Access configuration

5.1. Overview

The Java Client adapter will look into the local.bluenog.properties and the bluenog.properties files for the settings required for the connections it needs. It will need the information about the location of the Repository and access to the Repository's JMS queue for change update notification. The following settings are checked.

```
#----- Repository configuration for content access-----#
hippo.client.filespath=/files/default.preview
hippo.client.host=localhost
hippo.client.port=8080
hippo.client.username=root
hippo.client.password=password
hippo.client.realm=default realm
hippo.client.namespace=default

hippo.client.jms.factory=TCPTopicConnectionFactory
hippo.client.jms.topic=jms/repositoryTopic
```

```
hippo.client.jms.username=admin
hippo.client.jms.password=openjms
hippo.client.jms.reconnect.delay=1000
```