JBoss AS 6.0 JCA Guide

JCA with JBoss Application Server 6

by JBoss JCA Team

Part I. Introduction

JCA Introduction

1.1. JCA Overview

The Java Connector Architecture (JCA) defines a standard architecture for connecting the Java EE platform to heterogeneous Enterprise Information Systems (EIS). Examples of EISs include Enterprise Resource Planning (ERP), mainframe transaction processing (TP), databases and messaging systems.

The connector architecture defines a set of scalable, secure, and transactional mechanisms that enable the integration of EISs with application servers and enterprise applications.

The connector architecture also defines a Common Client Interface (CCI) for EIS access. The CCI defines a client API for interacting with heterogeneous EISs.

The connector architecture enables an EIS vendor to provide a standard resource adapter for its EIS. A resource adapter is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application. The resource adapter serves as a protocol adapter that allows any arbitrary EIS communication protocol to be used for connectivity. An application server vendor extends its system once to support the connector architecture and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard resource adapter which has the capability to plug in to any application server that supports the connector architecture.

1.2. What's New

New feature:

- How the JCA layer registers XA datasource for XA Resource Recovery with the JBoss TS project. See *Section 4.2, “XA Recovery in the JCA layer”*.

Part II. Configuration

Use Alternative Databases with JBoss AS

2.1. How to Use Alternative Databases

JBoss utilizes the Hypersonic database as its default database. While this is good for development and prototyping, you or your company will probably require another database to be used for production. This chapter covers configuring JBoss AS to use alternative databases. We cover the procedures for all officially supported databases on the JBoss Application Server. They include: MySQL 5.0, PostgreSQL 8.1, Oracle 9i and 10g R2, DB2 7.2 and 8, Sybase ASE 12.5, as well as MS SQL 2005.

Please note that in this chapter, we explain how to use alternative databases to support all services in JBoss AS. This includes all the system level services such as EJB and JMS. For individual applications (e.g., WAR or EAR) deployed in JBoss AS, you can still use any backend database by setting up the appropriate data source connection.

We assume that you have already installed the external database server, and have it running. You should create an empty database named `jboss`, accessible via the username / password pair `jbossuser / jbosspass`. The `jboss` database is used to store JBoss AS internal data -- JBoss AS will automatically create tables and data in it.

2.2. Install JDBC Drivers

For the JBoss Application Server and our applications to use the external database, we also need to install the database's JDBC driver. The JDBC driver is a JAR file, which you'll need to copy into your JBoss AS's `<JBoss_Home>/server/all/lib` directory. Replace `all` with the server configuration you are using if needed. This file is loaded when JBoss starts up. So if you have the JBoss AS running, you'll need to shut down and restart. The availability of JDBC drivers for different databases are as follows.

- IBM DB2 JDBC drivers can be downloaded from the IBM web site *http://www-306.ibm.com/software/data/db2/java/*.
- Sybase JDBC drivers can be downloaded from the Sybase jConnect product page *http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect*
- MS SQL Server JDBC drivers can be downloaded from the MSDN web site *http://msdn.microsoft.com/data/jdbc/*.

2.2.1. Special notes on Sybase

Some of the services in JBoss uses null values for the default tables that are created. Sybase Adaptive Server should be configured to allow nulls by default.

```
sp_dboption db_name, "allow nulls by default", true
```

Refer the sybase manuals for more options.

Enable JAVA services. To use any java service like JMS, CMP, timers etc. configured with Sybase, java should be enabled on Sybase Adaptive Server. To do this use:

```
sp_configure "enable java",1
```

Refer to the sybase manuals for more information.

If java is not enabled you might see this exception being thrown when you try to use any of the above services.

```
com.sybase.jdbc2.jdbc.SybSQLException: Cannot run this command because Java
 services are not enabled. A user with System Administrator (SA) role must
 reconfigure the system to enable Java
```

CMP Configuration. To use Container Managed Persistence for user defined Java objects with Sybase Adaptive Server Enterprise the java classes should be installed in the database. The system table 'sysxtypes' contains one row for each extended, Java-SQL datatype. This table is only used for Adaptive Servers enabled for Java. Install java classes using the installjava program.

```
installjava -f <jar-file-name> -S<sybase-server> -U<super-user>
 -P<super-pass> -D<db-name>
```

Refer the installjava manual in Sybase for more options.

Installing Java Classes

1. You have to be a super-user with required privileges to install java classes.
2. The jar file you are trying to install should be created without compression.
3. Java classes that you install and use in the server must be compiled with JDK 1.2.2. If you compile a class with a later JDK, you will be able to install it in the server using the installjava utility, but you will get a java.lang.ClassFormatError exception when you attempt to use the class. This is because Sybase Adaptive Server uses an older JVM internally, and hence requires the java classes to be compiled with the same.

2.2.2. Configuring JDBC DataSources

Rather than configuring the connection manager factory related MBeans discussed in the previous section via a mbean services deployment descriptor, JBoss provides a

simplified datasource centric descriptor. This is transformed into the standard `jboss-service.xml` MBean services deployment descriptor using a XSL transform applied by the `org.jboss.deployment.XSLSubDeployer` included in the `jboss-jca.sar` deployment. The simplified configuration descriptor is deployed the same as other deployable components. The descriptor must be named using a `*-ds.xml` pattern in order to be recognized by the `XSLSubDeployer`.

The schema for the top-level datasource elements of the `*-ds.xml` configuration deployment file is shown in *Figure 2.1, "The simplified JCA DataSource configuration descriptor top-level schema elements"*.

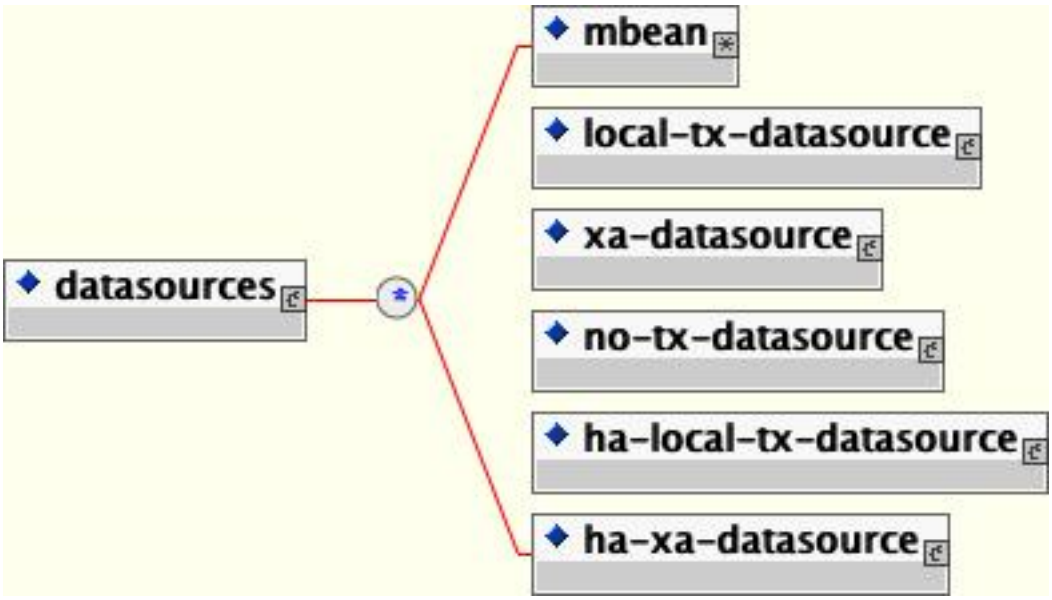

Figure 2.1. The simplified JCA DataSource configuration descriptor top-level schema elements

Multiple datasource configurations may be specified in a configuration deployment file. The child elements of the datasources root are:

- **mbean**: Any number mbean elements may be specified to define MBean services that should be included in the `jboss-service.xml` descriptor that results from the transformation. This may be used to configure services used by the datasources.
- **no-tx-datasource**: This element is used to specify the (`org.jboss.resource.connectionmanager`) `NoTxConnectionManager` service configuration. `NoTxConnectionManager` is a JCA connection manager with no transaction support. The `no-tx-datasource` child element schema is given in *Figure 2.2, "The non-transactional DataSource configuration schema"*.
- **local-tx-datasource**: This element is used to specify the (`org.jboss.resource.connectionmanager`) `LocalTxConnectionManager` service

configuration. `LocalTxConnectionManager` implements a `ConnectionEventListener` that implements `XAResource` to manage transactions through the transaction manager. To ensure that all work in a local transaction occurs over the same `ManagedConnection`, it includes a xid to `ManagedConnection` map. When a Connection is requested or a transaction started with a connection handle in use, it checks to see if a `ManagedConnection` already exists enrolled in the global transaction and uses it if found. Otherwise, a free `ManagedConnection` has its `LocalTransaction` started and is used. The `local-tx-datasource` child element schema is given in *Figure 2.3, "The non-XA DataSource configuration schema"*

- **xa-datasource**: This element is used to specify the (`org.jboss.resource.connectionmanager`) `XATxConnectionManager` service configuration. `XATxConnectionManager` implements a `ConnectionEventListener` that obtains the `XAResource` to manage transactions through the transaction manager from the adaptor `ManagedConnection`. To ensure that all work in a local transaction occurs over the same `ManagedConnection`, it includes a xid to `ManagedConnection` map. When a `Connection` is requested or a transaction started with a connection handle in use, it checks to see if a `ManagedConnection` already exists enrolled in the global transaction and uses it if found. Otherwise, a free `ManagedConnection` has its `LocalTransaction` started and is used. The `xa-datasource` child element schema is given in *Figure 2.4, "The XA DataSource configuration schema"*.

- **ha-local-tx-datasource**: This element is identical to `local-tx-datasource`, with the addition of the experimental datasource failover capability allowing JBoss to failover to an alternate database in the event of a database failure.

- **ha-xa-datasource**: This element is identical to `xa-datasource`, with the addition of the experimental datasource failover capability allowing JBoss to failover to an alternate database in the event of a database failure.

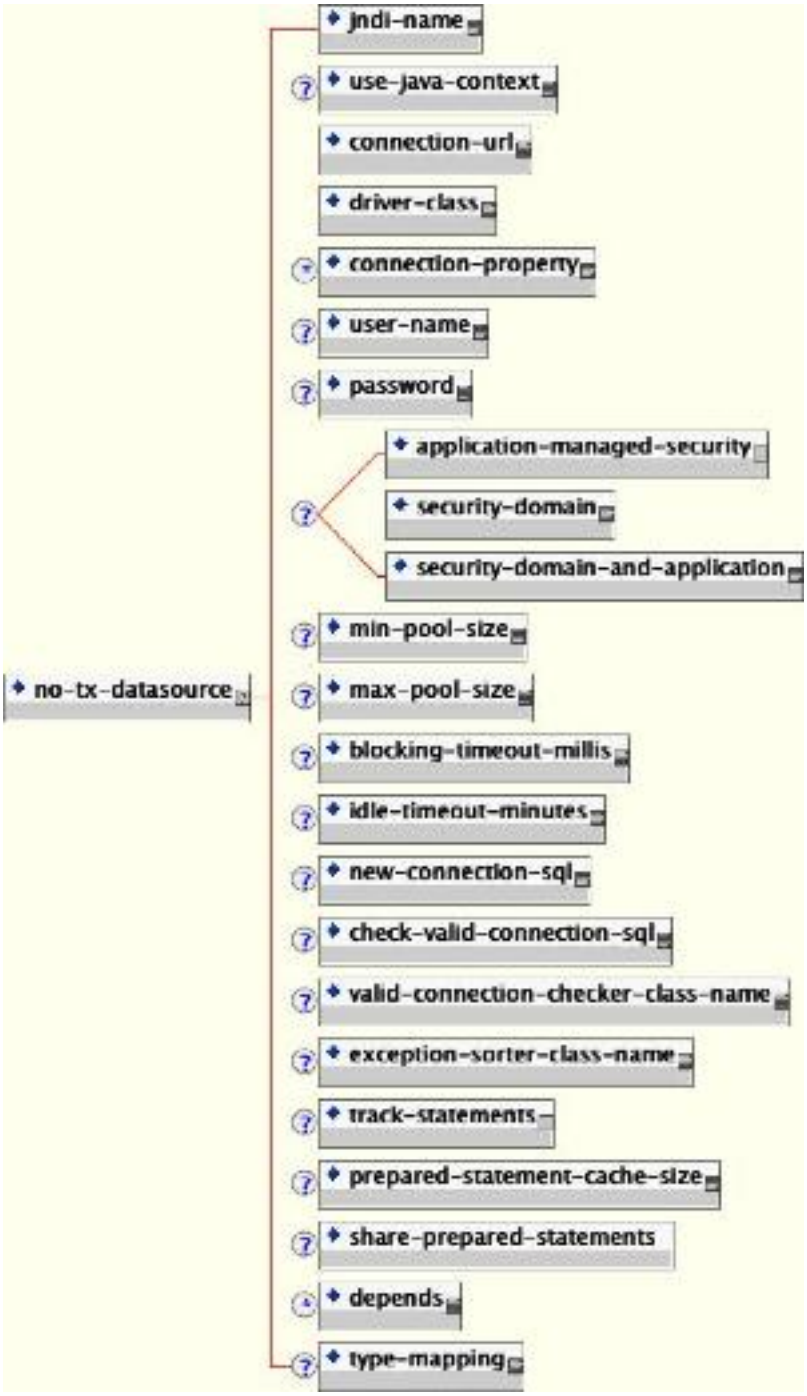

Figure 2.2. The non-transactional DataSource configuration schema

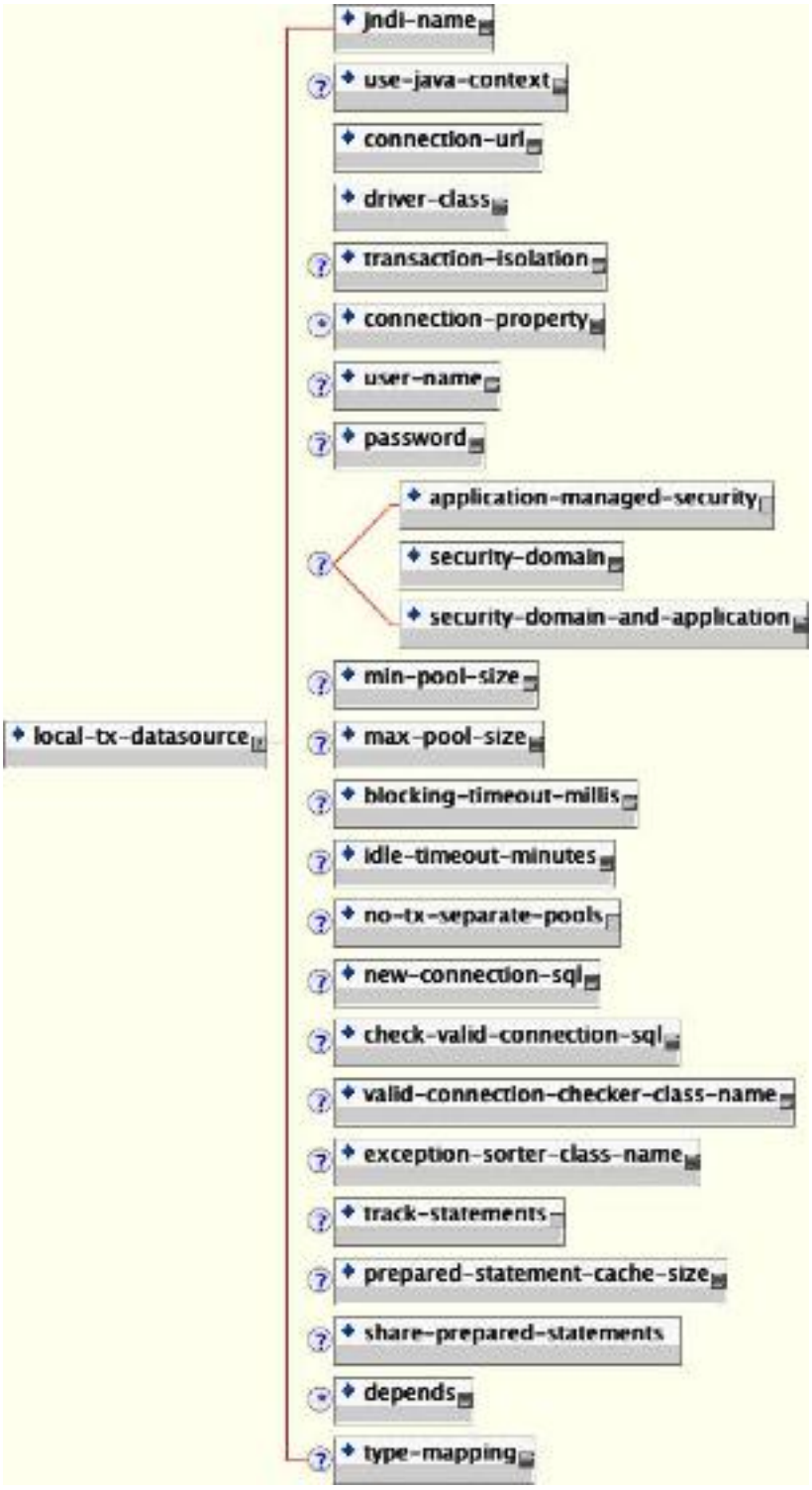

Figure 2.3. The non-XA DataSource configuration schema

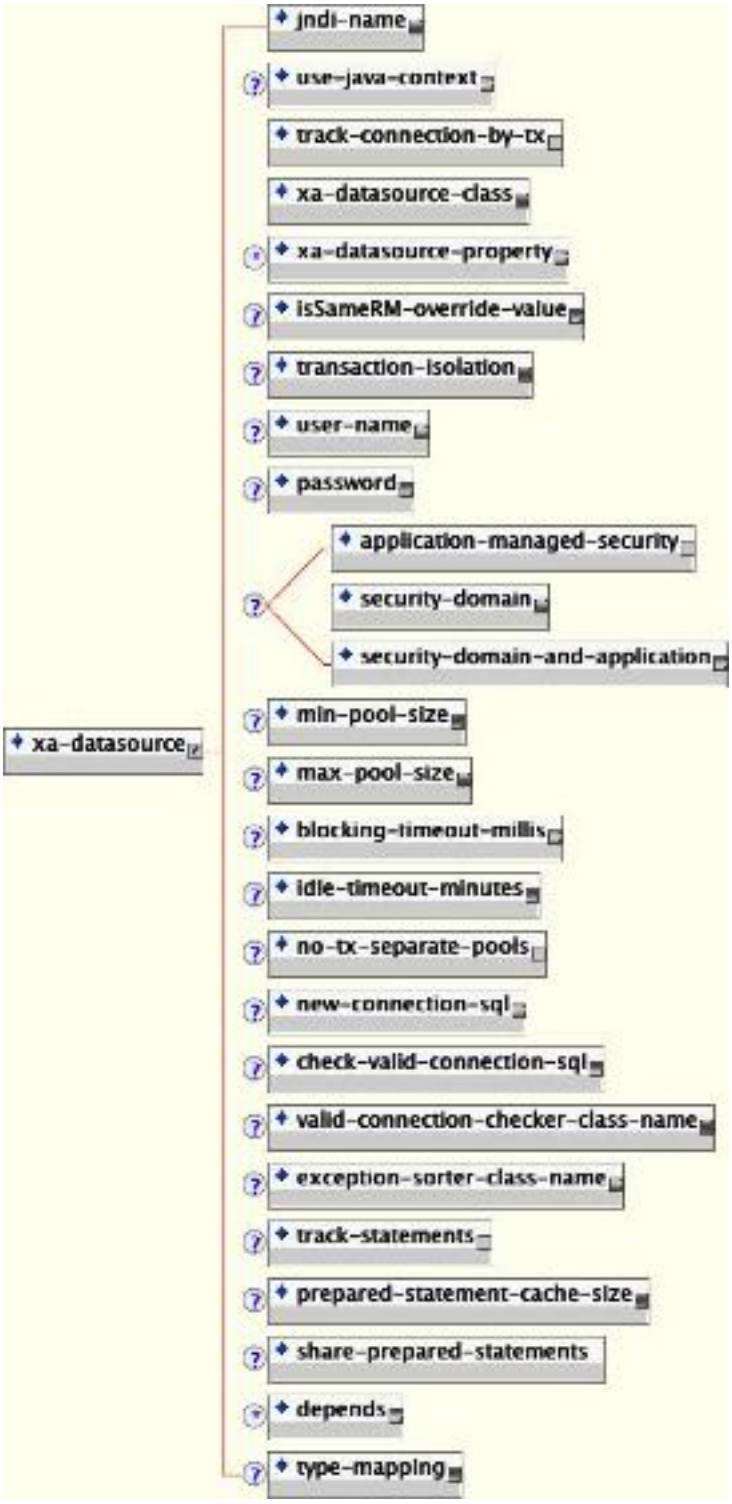

Figure 2.4. The XA DataSource configuration schema

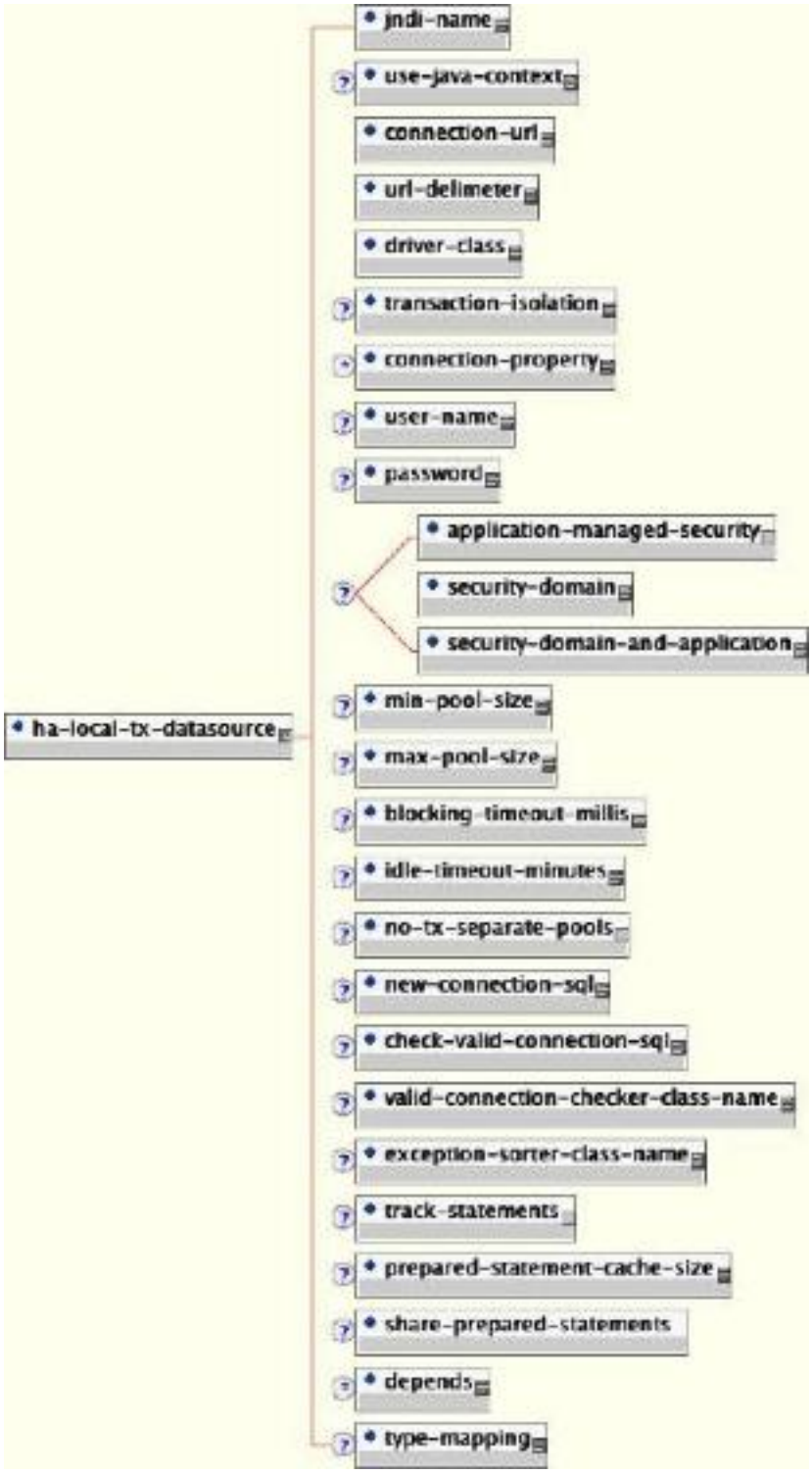

Figure 2.5. The schema for the experimental non-XA DataSource with failover

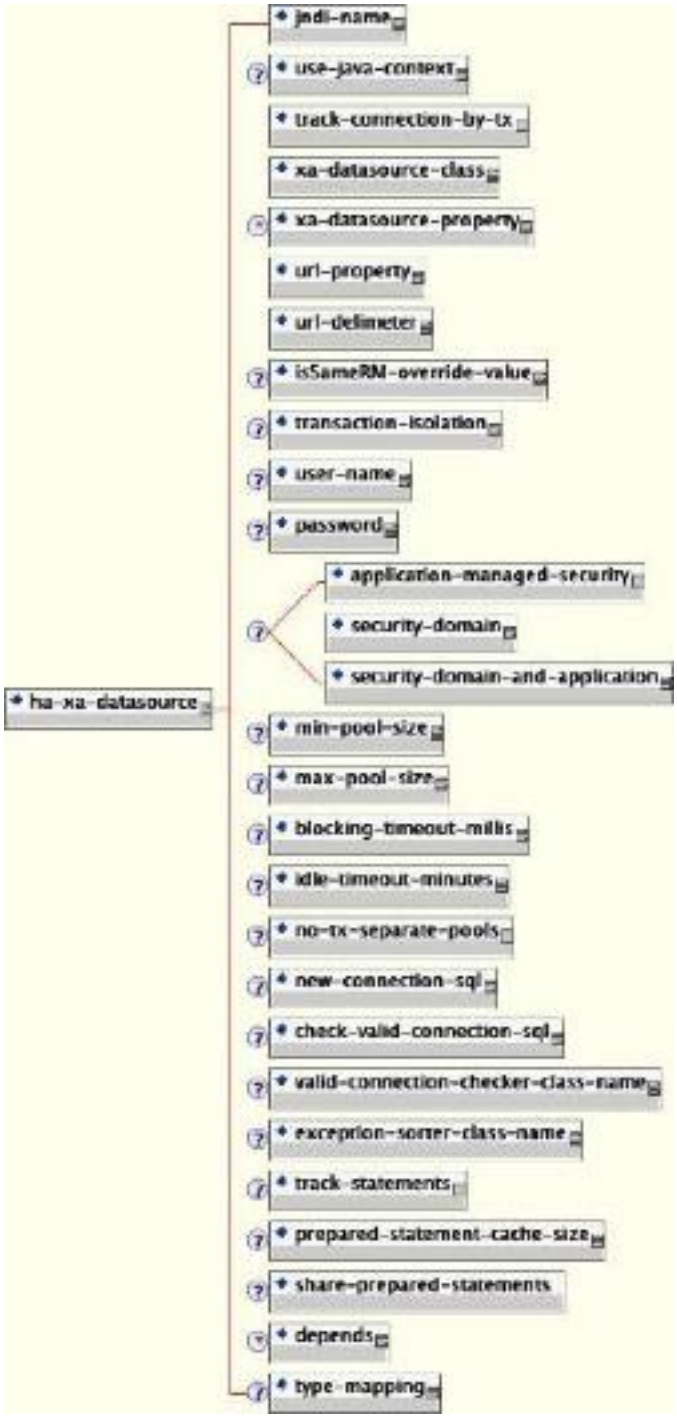

Figure 2.6. The schema for the experimental XA Datasource with failover

Elements that are common to all datasources include:

- **jndi-name**: The JNDI name under which the `DataSource` wrapper will be bound. Note that this name is relative to the `java:/` context, unless `use-java-context` is set to false. `DataSource` wrappers are not usable outside of the server VM, so they are normally bound under the `java:/`, which isn't shared outside the local VM.

- **use-java-context**: If this is set to false the the datasource will be bound in the global JNDI context rather than the `java:` context.
- **user-name**: This element specifies the default username used when creating a new connection. The actual username may be overridden by the application code `getConnection` parameters or the connection creation context JAAS Subject.
- **password**: This element specifies the default password used when creating a new connection. The actual password may be overridden by the application code `getConnection` parameters or the connection creation context JAAS Subject.
- **application-managed-security**: Specifying this element indicates that connections in the pool should be distinguished by application code supplied parameters, such as from `getConnection(user, pw)`.
- **security-domain**: Specifying this element indicates that connections in the pool should be distinguished by JAAS Subject based information. The content of the `security-domain` is the name of the JAAS security manager that will handle authentication. This name correlates to the JAAS `login-config.xml` descriptor `application-policy/name` attribute.
- **security-domain-and-application**: Specifying this element indicates that connections in the pool should be distinguished both by application code supplied parameters and JAAS Subject based information. The content of the `security-domain` is the name of the JAAS security manager that will handle authentication. This name correlates to the JAAS `login-config.xml` descriptor `application-policy/name` attribute.
- **min-pool-size**: This element specifies the minimum number of connections a pool should hold. These pool instances are not created until an initial request for a connection is made. This default to 0.
- **max-pool-size**: This element specifies the maximum number of connections for a pool. No more than the `max-pool-size` number of connections will be created in a pool. This defaults to 20.
- **blocking-timeout-millis**: This element specifies the maximum time in milliseconds to block while waiting for a connection before throwing an exception. Note that this blocks only while waiting for a permit for a connection, and will never throw an exception if creating a new connection takes an inordinately long time. The default is 5000.
- **idle-timeout-minutes**: This element specifies the maximum time in minutes a connection may be idle before being closed. The actual maximum time depends also on the `IdleRemover` scan time, which is 1/2 the smallest idle-timeout-minutes of any pool.
- **new-connection-sql**: This is a SQL statement that should be executed when a new connection is created. This can be used to configure a connection with database specific settings not configurable via connection properties.
- **check-valid-connection-sql**: This is a SQL statement that should be run on a connection before it is returned from the pool to test its validity to test for stale pool connections. An example statement could be: `select count(*) from x`.

- **exception-sorter-class-name**: This specifies a class that implements the `org.jboss.resource.adapter.jdbc.ExceptionSorter` interface to examine database exceptions to determine whether or not the exception indicates a connection error. Current implementations include:
  - org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter
  - org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter
  - org.jboss.resource.adapter.jdbc.vendor.SybaseExceptionSorter
  - org.jboss.resource.adapter.jdbc.vendor.InformixExceptionSorte
- **valid-connection-checker-class-name**: This specifies a class that implements the `org.jboss.resource.adapter.jdbc.ValidConnectionChecker` interface to provide a `SQLException isValidConnection(Connection e)` method that is called with a connection that is to be returned from the pool to test its validity. This overrides the `check-valid-connection-sql` when present. The only provided implementation is `org.jboss.resource.adapter.jdbc.vendor.OracleValidConnectionChecker`.
- **track-statements**: This boolean element specifies whether to check for unclosed statements when a connection is returned to the pool. If true, a warning message is issued for each unclosed statement. If the log4j category `org.jboss.resource.adapter.jdbc.WrappedConnection` has trace level enabled, a stack trace of the connection close call is logged as well. This is a debug feature that can be turned off in production.
- **prepared-statement-cache-size**: This element specifies the number of prepared statements per connection in an LRU cache, which is keyed by the SQL query. Setting this to zero disables the cache.
- **depends**: The `depends` element specifies the JMX `ObjectName` string of a service that the connection manager services depend on. The connection manager service will not be started until the dependent services have been started.
- **type-mapping**: This element declares a default type mapping for this datasource. The type mapping should match a `type-mapping/name` element from `standardjbosscmp-jdbc.xml`.

Additional common child elements for both `no-tx-datasource` and `local-tx-datasource` include:

- **connection-url**: This is the JDBC driver connection URL string, for example, `jdbc:hsqldb:hsql://localhost:1701`.
- **driver-class**: This is the fully qualified name of the JDBC driver class, for example, `org.hsqldb.jdbcDriver`.
- **connection-property**: The `connection-property` element allows you to pass in arbitrary connection properties to the `java.sql.Driver.connect(url, props)` method. Each `connection-property` specifies a string name/value pair with the property name coming from the name attribute and the value coming from the element content.

Elements in common to the `local-tx-datasource` and `xa-datasource` are:

- **transaction-isolation**: This element specifies the `java.sql.Connection` transaction isolation level to use. The constants defined in the Connection interface are the possible element content values and include:
  - TRANSACTION_READ_UNCOMMITTED
  - TRANSACTION_READ_COMMITTED
  - TRANSACTION_REPEATABLE_READ
  - TRANSACTION_SERIALIZABLE
  - TRANSACTION_NONE
- **no-tx-separate-pools**: The presence of this element indicates that two connection pools are required to isolate connections used with JTA transaction from those used without a JTA transaction. The pools are lazily constructed on first use. Its use case is for Oracle (and possibly other vendors) XA implementations that don't like using an XA connection with and without a JTA transaction.

The unique `xa-datasource` child elements are:

- **track-connection-by-tx**: Specifying a true value for this element makes the connection manager keep an xid to connection map and only put the connection back in the pool when the transaction completes and all the connection handles are closed or disassociated (by the method calls returning). As a side effect, we never suspend and resume the xid on the connection's `XAResource`. This is the same connection tracking behavior used for local transactions.

  The XA spec implies that any connection may be enrolled in any transaction using any xid for that transaction at any time from any thread (suspending other transactions if necessary). The original JCA implementation assumed this and aggressively delisted connections and put them back in the pool as soon as control left the EJB they were used in or handles were closed. Since some other transaction could be using the connection the next time work needed to be done on the original transaction, there is no way to get the original connection back. It turns out that most `XADataSource` driver vendors do not support this, and require that all work done under a particular xid go through the same connection.
- **xa-datasource-class**: The fully qualified name of the `javax.sql.XADataSource` implementation class, for example, `com.informix.jdbcx.IfxXADataSource`.
- **xa-datasource-property**: The `xa-datasource-property` element allows for specification of the properties to assign to the `XADataSource` implementation class. Each property is identified by the name attribute and the property value is given by the `xa-datasource-property` element content. The property is mapped onto the `XADataSource` implementation by looking for a JavaBeans style getter method for the property name. If found, the value of the property is set using the JavaBeans setter with the element text translated to the true property type using the `java.beans.PropertyEditor` for the type.

- **isSameRM-override-value**: A boolean flag that allows one to override the behavior of the `javax.transaction.xa.XAResource.isSameRM(XAResource xaRes)` method behavior on the XA managed connection. If specified, this value is used unconditionally as the `isSameRM(xaRes)` return value regardless of the `xaRes` parameter.

The failover options common to `ha-xa-datasource` and `ha-local-tx-datasource` are:

- **url-delimeter**: This element specifies a character used to separate multiple JDBC URLs.
- **url-property**: In the case of XA datasources, this property specifies the name of the `xa-datasource-property` that contains the list of JDBC URLs to use.

2.3. Creating a DataSource for the External Database

JBoss AS connects to relational databases via datasources. These datasource definitions can be found in the `<JBoss_Home>/server/all/deploy` directory. The datasource definitions are deployable just like WAR and EAR files. The datasource files can be recognized by looking for the XML files that end in `*-ds.xml`.

> **Datasource definition files**
>
> The datasource definition files for all supported external databases can be found in the `<JBoss_Home>/docs/examples/jca` directory.

- MySQL: `mysql-ds.xml`
- PostgreSQL: `postgres-ds.xml`
- Oracle: `oracle-ds.xml`
- DB2: `db2-ds.xml`
- Sybase: `sybase-ds.xml`
- MS SQL Server: `mssql-ds.xml`

The following code snippet shows the `mysql-ds.xml` file as an example. All the other `*-ds.xml` files are very similiar. You will need to change the `connection-url`, as well as the `user-name` / `password`, to fit your own database server installation.

```
<datasources>
<local-tx-datasource>
<jndi-name>MySqlDS</jndi-name>
```

```
<connection-url>jdbc:mysql://localhost:3306/jboss</connection-url>
<driver-class>com.mysql.jdbc.Driver</driver-class>
<user-name>jbossuser</user-name>
<password>jbosspass</password>
<exception-sorter-class-name>
org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter
</exception-sorter-class-name>
<!-- should only be used on drivers after 3.22.1 with "ping" support
<valid-connection-checker-class-name>
org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker
</valid-connection-checker-class-name>
-->
<!-- sql to call when connection is created
<new-connection-sql>some arbitrary sql</new-connection-sql>
-->
<!-- sql to call on an existing pooled connection when it is obtained from pool -
 MySQLValidConnectionChecker is preferred for newer drivers
<check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>
 -->

<!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (optional) -->
   <metadata>
 <type-mapping>mySQL</type-mapping>
 </metadata>
 </local-tx-datasource>

</datasources>
```

Once you customized the `*-ds.xml` file to connect to your external database, you need to copy it to the `<JBoss_Home>/server/all/deploy` directory. The database connection is now available through the JNDI name specified in the `*-ds.xml` file.

2.4. Common configuration for DataSources and ConnectionFactorys

2.4.1. General

- *<mbean>* - a standard jboss mbean deployment
- *<depends>* - the ObjectName of an MBean service this ConnectionFactory or DataSource deployment depends upon
- *<jndi-name>* - the jndi name where it is bound. This is prefixed with java by default:

- <*use-java-context*> - set this to false to drop the java: context from the jndi name

2.4.2. XA

<*xa-resource-timeout*> - the number of seconds passed to

```
XAResource.setTranasctionTimeout()
```

when not zero. This feature is available on JBoss AS 4.0.3 and above.

2.4.3. Security parameters

JCA Login Modules - are used to inject security configuration into the connection when configured

- *nothing* - uses the user/password specified in `-ds.xml` for DataSources or the `getConnection/createConnection` method without a `user/password` (the default).
- <*application-managed-security*> - uses the user/password passed on the `getConnection` or `createConnection` request by the application.
- <*security-domain*> - uses the identified login module configured in `conf/login-module.xml`.
- <*security-domain-and-application*> - uses the identified login module configured in `conf/login-module.xml` and other connection request information supplied by the application, e.g. queue or topic in JMS.

2.4.3.1. Pooling parameters

- <*no-tx-separate-pools*> - whether separate subpools should be created for connections inside and outside JTA transactions (default false).
- <*min-pool-size*> - the minimum number of connections in the pool (default 0 - zero)
- <*max-pool-size*> - the maximum number of connections in the pool (default 20)
- <*blocking-timeout-millis*> - the length of time to wait for a connection to become available when all the connections are checked out (default 5000 == 5 seconds, from 3.2.4 it is 30000 == 30 seconds)
- <*idle-timeout-minutes*> - the number of minutes after which unused connections are closed (default 15 minutes)
- <*track-connection-by-tx*> - whether the connection should be *"locked"* to the transaction, returning it to the pool at the end of the transaction; in pre-JBoss-5.x releases the default value for Local connection factories is true and false for XA; since JBoss-5.x the default value is true for both Local and XA and the element is deprecated.

- <*interleaving/*> - enables interleaving for XA connection factories (this feature was added in JBoss-5.x)
- <*prefill*> - whether to attempt to prefill the connection pool to the minimum number of connections. NOTE: only supporting pools (OnePool) support this feature. A warning can be found in the logs if the pool does not support this. This feature is available in JBoss 4.0.5 and above.
- <*background-validation*> - In JBoss 4.0.5, background connection validation was added to reduce the overall load on the RDBMS system when validating a connection. When using this feature, JBoss will attempt to validate the current connections in the pool as a seperate thread (ConnectionValidator).
- <*background-validation-minutes*> - The interval, in minutes, that the ConnectionValidator will run. NOTE: It is prudent to set this value to something greater or less than the <*idle-timeout-minutes*>
- <*use-fast-fail*> - Whether or not to continue to attempt to acquire a connection from the pool even if the nth attempt has failed. False by default. This is to address performance issues where SQL validation may take significant time and resources to execute.

2.4.3.2. Security and Pooling

Unless the ResourceAdapter has <*reauthentication-support*> using multiple security identities will create subpools for each identity.

> **Note**
>
> The min and max pool size are per subpool so be careful with these parameters if you have lots of identities.

2.5. Change Database for the JMS Services

The JMS service in the JBoss AS uses relational databases to persist its messages. For improved performance, we should change the JMS service to take advantage of the external database. To do that, we need to replace the file `<JBoss_Home>/server/all/deploy/jms-singleton/hsqldb-jdbc2-service.xml` with a file in `<JBoss_Home>/docs/examples/jms/` depending on your external database. Notice that if you are using the `default` server profile, the file path is `<JBoss_Home>/server/default/deploy/jms/hsqldb-jdbc2-service.xml`.

- MySQL: `mysql-jdbc2-service.xml`
- PostgreSQL: `postgres-jdbc2-service.xml`
- Oracle: `oracle-jdbc2-service.xml`

- DB2: `db2-jdbc2-service.xml`
- Sybase: `sybase-jdbc2-service.xml`
- MS SQL Server: `mssql-jdbc2-service.xml`

> **What about the hsqldb-jdbc-state-service.xml file?**
>
> Despite its name, the `hsqldb-jdbc-state-service.xml` file applies to all databases. So, there is no need to use a special `jdbc-state-service.xml` for each database.

2.6. Support Foreign Keys in CMP Services

Next, we need to go change the `<JBoss_Home>/server/all/conf/standardjbosscmp-jdbc.xml` file so that the `fk-constraint` property is `true`. That is needed for all external databases we support on the JBoss Application Server. This file configures the database connection settings for the EJB2 CMP beans deployed in the JBoss AS.

```
<fk-constraint>true</fk-constraint>
```

2.7. Specify Database Dialect for Java Persistence API

The Java Persistence API (JPA) entity manager can save EJB3 entity beans to any backend database. Hibernate provides the JPA implementation in JBoss AS. Hibernate has a dialect auto-detection mechanism that works for most databases including the dialects for databases referenced in this appendix which are listed below. If a specific dialect is needed for alternative databases, you can configure the database dialect in the `<JBoss_Home>/server/all/deploy/ejb3.deployer/META-INF/persistence.properties` file. You need to un-comment the `hibernate.dialect` property and change its value to the following based on the database you setup. For a complete list of dialects, refer to the Hibernate Reference Guide, Chapter 3, Section 4.1 SQL Dialects.

- Oracle 9i: org.hibernate.dialect.Oracle9iDialect
- Oracle 10g: org.hibernate.dialect.Oracle10gDialect
- Microsoft SQL Server 2005: org.hibernate.dialect.SQLServerDialect
- PostgresSQL 8.1: org.hibernate.dialect.PostgreSQLDialect
- MySQL 5.0: org.hibernate.dialect.MySQL5Dialect
- DB2 8.0: org.hibernate.dialect.DB2Dialect

- Sybase ASE 12.5: org.hibernate.dialect.SybaseDialect

DB2 7.2 with Universal JDBC Driver (Type 4)

Large Objects (LOBs) are supported only with DB2 Version 8 servers and above with the universal JDBC driver. Hence JMS services which stores messages as BLOBS and Timer services which uses BLOB fields for storing objects do not work with the JDBC Type 4 driver and DB2 7.2.

DB2 7.2 with JDBC Type 2 driver

All JBoss services work with the JDBC Type 2 driver and DB2 Version 7.2 servers.

2.8. Change Other JBoss AS Services to Use the External Database

Besides JMS, CMP, and JPA, we still need to hook up the rest of JBoss services with the external database. There are two ways to do it. One is easy but inflexible. The other is flexible but requires more steps. Now, let's discuss those two approaches respectively.

2.8.1. The Easy Way

The easy way is just to change the JNDI name for the external database to `DefaultDS`. Most JBoss services are hard-wired to use the `DefaultDS` by default. So, by changing the datasource name, we do not need to change the configuration for each service individually.

To change the JNDI name, just open the `*-ds.xml` file for your external database, and change the value of the `jndi-name` property to `DefaultDS`. For instance, in `mysql-ds.xml`, you'd change MySqlDS to DefaultDS and so on. You will need to remove the `<JBoss_Home>/server/all/deploy/hsqldb-ds.xml` file after you are done to avoid duplicated `DefaultDS` definition.

In the `jms/*-jdbc2-service.xml` file, you should also change the datasource name in the `depends` tag for the `PersistenceManagers` MBean to `DefaultDS`. For instance, for `mysql-jdbc2-service.xml` file, we change the `MySqlDS` to `DefaultDS`.

The easy way is just to change the JNDI name for the external database to `DefaultDS`. Most JBoss services are hard-wired to use the `DefaultDS` by default. So, by changing the datasource name, we do not need to change the configuration for each service individually.

To change the JNDI name, just open the `*-ds.xml` file for your external database, and change the value of the `jndi-name` property to `DefaultDS`. For instance, in `mysql-ds.xml`, you'd change `MySqlDS` to `DefaultDS` and so on. You will need to remove the `<JBoss_Home>/server/all/deploy/hsqldb-ds.xml` file after you are done to avoid duplicated `DefaultDS` definition.

In the `jms/*-jdbc2-service.xml` file, you should also change the datasource name in the `depends` tag for the `PersistenceManagers` MBean to `DefaultDS`. For instance, for `mysql-jdbc2-service.xml` file, we change the `MySqlDS` to `DefaultDS`.

```
.. ...
<mbean code="org.jboss.mq.pm.jdbc2.PersistenceManager"
      name="jboss.mq:service=PersistenceManager"%gt;
  <depends optional-attribute-name="ConnectionManager">
    jboss.jca:service=DataSourceBinding,name=DefaultDS
  </depends>
... ...
```

2.8.2. The More Flexible Way

Changing the external datasource to `DefaultDS` is convenient. But if you have applications that assume the `DefaultDS` always points to the factory-default HSQL DB, that approach could break your application. Also, changing `DefaultDS` destination forces all JBoss services to use the external database. What if you want to use the external database only on some services?

A safer and more flexible way to hook up JBoss AS services with the external datasource is to manually change the `DefaultDS` in all standard JBoss services to the datasource JNDI name defined in your `*-ds.xml` file (e.g., the `MySqlDS` in `mysql-ds.xml` etc.). Below is a complete list of files that contain `DefaultDS`. You can update them all to use the external database on all JBoss services or update some of them to use different combination of datasources for different services.

- `<JBoss_Home>/server/all/conf/login-config.xml`: This file is used in Java EE container managed security services.
- `<JBoss_Home>/server/all/conf/standardjbosscmp-jdbc.xml`: This file configures the CMP beans in the EJB container.
- `<JBoss_Home>/server/all/deploy/ejb-deployer.xml`: This file configures the JBoss EJB deployer.
- `<JBoss_Home>/server/all/deploy/schedule-manager-service.xml`: This file configures the EJB timer services.
- `<JBoss_Home>/server/all/deploy/snmp-adaptor.sar/attributes.xml`: This file is used by the SNMP service.
- `<JBoss_Home>/server/all/deploy/juddi-service.sar/META-INF/jboss-service.xml`: This file configures the UUDI service.
- `<JBoss_Home>/server/all/deploy/juddi-service.sar/juddi.war/WEB-INF/jboss-web.xml`: This file configures the UUDI service.

- `<JBoss_Home>/server/all/deploy/juddi-service.sar/juddi.war/WEB-INF/juddi.properties`: This file configures the UUDI service.
- `<JBoss_Home>/server/all/deploy/uuid-key-generator.sar/META-INF/jboss-service.xml`: This file configures the UUDI service.
- `<JBoss_Home>/server/all/jms/hsqldb-jdbc-state-service.xml` and `<JBoss_Home>/server/all/deploy-hasingleton/jms/hsqldb-jdbc-state-service.xml`: Those files configure the JMS persistence service as we discussed earlier.

2.9. A Special Note About Oracle DataBases

In our setup discussed in this chapter, we rely on the JBoss AS to automatically create needed tables in the external database upon server startup. That works most of the time. But for databases like Oracle, there might be some minor issues if you try to use the same database server to back more than one JBoss AS instance.

The Oracle database creates tables of the form `schemaname.tablename`. The `TIMERS` and `HILOSEQUENCES` tables needed by JBoss AS would not get created on a schema if the table already exists on a different schema. To work around this issue, you need to edit the `<JBoss_Home>/server/all/deploy/ejb-deployer.xml` file to change the table name from `TIMERS` to something like `schemaname2.tablename`.

```
<mbean code="org.jboss.ejb.txtimer.DatabasePersistencePolicy"
name="jboss.ejb:service=EJBTimerService,persistencePolicy=database">
<!-- DataSourceBinding ObjectName -->
<depends optional-attribute-name="DataSource">
 jboss.jca:service=DataSourceBinding,name=DefaultDS
</depends>
<!-- The plugin that handles database persistence -->
<attribute name="DatabasePersistencePlugin">
org.jboss.ejb.txtimer.GeneralPurposeDatabasePersistencePlugin
</attribute>
<!-- The timers table name -->
<attribute name="TimersTable">TIMERS</attribute>
</mbean>
```

Similarly, you need to change the `<JBoss_Home>/server/all/deploy/uuid-key-generator.sar/META-INF/jboss-service.xml` file to change the table name from `HILOSEQUENCES` to something like `schemaname2.tablename` as well.

```
<!-- HiLoKeyGeneratorFactory -->
<mbean code="org.jboss.ejb.plugins.keygenerator.hilo.HiLoKeyGeneratorFactory"
name="jboss:service=KeyGeneratorFactory,type=HiLo">
```

```
<depends>jboss:service=TransactionManager</depends>

<!-- Attributes common to HiLo factory instances -->

<!-- DataSource JNDI name -->
<depends optional-attribute-name="DataSource">jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>

<!-- table name -->
<attribute name="TableName">HILOSEQUENCES</attribute>
```

2.10. DataSource configuration

DataSources are defined inside a <datasources> element.

- <no-tx-datasource> - a DataSource that does not take part in JTA transactions using a java.sql.Driver
- <local-tx-datasource> - a DataSource that does not support two phase commit using a java.sql.Driver
- <xa-datasource> - a DataSource that does support two phase commit using a javax.sql.XADataSource

2.11. Parameters specific for java.sql.Driver usage

- <connection-url> - the JDBC driver connection url string
- <driver-class> - the JDBC driver class implementing java.sql.Driver
- <connection-property> - used to configure the connections retrieved from the java.sql.Driver. For example:

```
<connection-property name="char.encoding">UTF-8</connection-property>
```

2.12. Parameters specific for javax.sql.XADataSource usage

- <xa-datasource-class> - the class implementing the XADataSource
- <xa-datasource-property> - properties used to configure the XADataSource. For example:

```
<xa-datasource-property name="IfxWAITTIME">10</xa-datasource-property>
```

```
<xa-datasource-property name="IfxIFXHOST">myhost.mydomain.com</xa-datasource-property>
<xa-datasource-property name="PortNumber">1557</xa-datasource-property>
<xa-datasource-property name="DatabaseName">mydb</xa-datasource-property>
<xa-datasource-property name="ServerName">myserver</xa-datasource-property>
```

- <isSameRM-override-value> - set to false to fix problems with Oracle
- <track-connection-by-tx/> - set to fix problems with Oracle (not necessarily in JBoss-5.x where it is enabled by default and the element is deprecated)
- <no-tx-separate-pools/> - Pool Transactional and non-Transactional connections separately. Using this option will cause your total pool size to be twice max-pool-size because two actual pools will be created. Used to fix problems with Oracle.

2.13. Common DataSource parameters

- <jndi-name> - the JNDI name under which the DataSource should be bound.
- <use-java-context> - A boolean indicating if the jndi-name should be prefixed with java: which causes the DataSource to only be accessible from within the jboss server vm. The default is true.
- <user-name> - the user name used when creating the connection (not used when security is configured)
- <password> - the password used when creating the connection (not used when security is configured)
- <transaction-isolation> - the default transaction isolation of the connection (unspecified means use the default provided by the database):
  - TRANSACTION_READ_UNCOMMITTED
  - TRANSACTION_READ_COMMITTED
  - TRANSACTION_REPEATABLE_READ
  - TRANSACTION_SERIALIZABLE
  - TRANSACTION_NONE
- <new-connection-sql> - an sql statement that is executed against each new connection. This can be used to set the connection schema, etc.
- <check-valid-connection-sql> - an sql statement that is executed before it is checked out from the pool to make sure it is still valid. If the sql fails, the connection is closed and new ones created.
- <valid-connection-checker-class-name> - a class that can check whether a connection is valid using a vendor specific mechanism

- <exception-sorter-class-name> - a class that looks at vendor specific messages to determine whether sql errors are fatal and thus the connection should be destroyed. If none specified, no errors will be treated as fatal.
- <track-statements> - (a) whether to monitor for unclosed Statements and ResultSets and issue warnings when the user forgets to close them (default nowarn)
- <prepared-statement-cache-size> - the number of prepared statements per connection to be kept open and reused in subsequent requests. They are stored in a LRU cache. The default is 0 (zero), meaning no cache.
- <share-prepared-statements> - (b) with prepared statement cache enabled whether two requests in the same transaction should return the same statement (from jboss-4.0.2 - default false).
- <set-tx-query-timeout> - whether to enable query timeout based on the length of time remaining until the transaction times out (default false - NOTE: This was NOT ported to 4.0.x until 4.0.3)
- <query-timeout> - a static configuration of the maximum of seconds before a query times out (since 4.0.3)
- <metadata/typemapping> - a pointer to the type mapping in conf/standardjbosscmp.xml (available from JBoss 4 and above)
- <validate-on-match> - Prior to JBoss 4.0.5, connection validation occurred when the JCA layer attempted to match a managed connection. With the addition of <background-validation> this is no longer required. Specifying <validate-on-match> forces the old behavior. NOTE: this is typically NOT used in conjunction with <background-validation>
- <prefill> - whether to attempt to prefill the connection pool to the minimum number of connections. NOTE: only supporting pools (OnePool) support this feature. A warning can be found in the logs if the pool does not support this. This feature will appear in JBoss 4.0.5.
- <background-validation> - In JBoss 4.0.5, background connection validation as been added to reduce the overall load on the RDBMS system when validating a connection. When using this feature, JBoss will attempt to validate the current connections in the pool is a seperate thread (ConnectionValidator). Default is False.
- <idle-timeout-minutes> - indicates the maximum time a connection may be idle before being closed. Default is 15 minutes.
- <background-validation-minutes> - The interval, in minutes, that the ConnectionValidator will run. Default is 10 minutes. NOTE: It is prudent to set this value to something greater or less than the <idle-timeout-minutes>
- <url-delimiter> - From JBoss5 database failover is part of the main datasource config
- <url-property> - From JBoss5 database failover is part of the main datasource config
- <url-selector-strategy-class-name> - From JBoss5 ONLY database failover is part of the main datasource config

- <stale-connection-checker-class-name> - An implementation of org.jboss.resource.adapter.jdbc.StateConnectionChecker that will decide whether SQLExceptions that notify of bad connections throw org.jboss.resource.adapter.jdbc.StateConnectionException (from JBoss5)

From JBoss AS 3.2.6 and above, `track-statements` has a new option:

```
<track-statements>nowarn</track-statements
```

This option closes Statements and ResultSets without a warning. It is also the new default value.

The purpose is to workaround questionable driver behavior where the driver applies auto-commit semantics to local transactions.

```
Connection c = dataSource.getConnection(); // auto-commit == false
PreparedStatement ps1 = c.prepareStatement(...);
ResultSet rs1 = ps1.executeQuery();
PreparedStatement ps2 = c.prepareStatement(...);
ResultSet rs2 = ps2.executeQuery();
```

Assuming the prepared statements are the same. For some drivers, ps2.executeQuery() will automatically close rs1 so we actually need two real prepared statements behind the scenes. This *should* only be for the auto-commit semantic where re-running the query starts a new transaction automatically. For drivers that follow the spec, you can set it to true to share the same real prepared statement.

2.14. Generic Datasource Sample

```
<datasources>
<local-tx-datasource>
<jndi-name>GenericDS</jndi-name>
<connection-url>[jdbc: url for use with Driver class]</connection-url>
<driver-class>[fully qualified class name of java.sql.Driver implementation]</driver-class>
<user-name>x</user-name>
<password>y</password>
<!-- you can include connection properties that will get passed in
the DriverManager.getConnection(props) call-->
<!-- look at your Driver docs to see what these might be -->
<connection-property name="char.encoding">UTF-8</connection-property>
<transaction-isolation>TRANSACTION_SERIALIZABLE</transaction-isolation>

<!--pooling parameters-->
```

```
<min-pool-size>5</min-pool-size>
<max-pool-size>100</max-pool-size>
<blocking-timeout-millis>5000</blocking-timeout-millis>
<idle-timeout-minutes>15</idle-timeout-minutes>
<!-- sql to call when connection is created
<new-connection-sql>some arbitrary sql</new-connection-sql>
-->

<!-- sql to call on an existing pooled connection when it is obtained from pool
<check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>
-->

<set-tx-query-timeout/>
<query-timeout>300</query-timeout> <!-- maximum of 5 minutes for queries -->

<!-- pooling criteria.  USE AT MOST ONE-->
<!--  If you don't use JAAS login modules or explicit login
getConnection(usr,pw) but rely on user/pw specified above,
don't specify anything here -->

<!-- If you supply the usr/pw from a JAAS login module -->
<security-domain>MyRealm</security-domain>

<!-- if your app supplies the usr/pw explicitly getConnection(usr, pw) -->
<application-managed-security/>

<!--Anonymous depends elements are copied verbatim into the ConnectionManager mbean
 config-->
<depends>myapp.service:service=DoSomethingService</depends>

</local-tx-datasource>

<!-- you can include regular mbean configurations like this one -->
<mbean code="org.jboss.tm.XidFactory"
name="jboss:service=XidFactory">
<attribute name="Pad">true</attribute>
</mbean>

<!-- Here's an xa example -->
<xa-datasource>
<jndi-name>GenericXADS</jndi-name>
<xa-datasource-class>[fully qualified name of class implementing javax.sql.XADataSource goes
 here]</xa-datasource-class>
```

```
<xa-datasource-property name="SomeProperty">SomePropertyValue</xa-datasource-property>
<xa-datasource-property name="SomeOtherProperty">SomeOtherValue</xa-datasource-property>

<user-name>x</user-name>
<password>y</password>
<transaction-isolation>TRANSACTION_SERIALIZABLE</transaction-isolation>

<!--pooling parameters-->
<min-pool-size>5</min-pool-size>
<max-pool-size>100</max-pool-size>
<blocking-timeout-millis>5000</blocking-timeout-millis>
<idle-timeout-minutes>15</idle-timeout-minutes>
<!-- sql to call when connection is created
<new-connection-sql>some arbitrary sql</new-connection-sql>
-->

<!-- sql to call on an existing pooled connection when it is obtained from pool
<check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>
-->

<!-- pooling criteria.  USE AT MOST ONE-->
<!--  If you don't use JAAS login modules or explicit login
getConnection(usr,pw) but rely on user/pw specified above,
don't specify anything here -->

<!-- If you supply the usr/pw from a JAAS login module -->
<security-domain/>

<!-- if your app supplies the usr/pw explicitly getConnection(usr, pw) -->
<application-managed-security/>

</xa-datasource>

</datasources>
```

2.15. Configuring a DataSource for remote usage

From JBoss-4.0.0 and above, there is support for accessing a DataSource from a remote client. The one change that is necessary for the client to be able to lookup the DataSource from JNDI is to specify use-java-context=false as shown here:

```
<datasources>
<local-tx-datasource>
<jndi-name>GenericDS</jndi-name>
<use-java-context>false</use-java-context>
<connection-url>...</connection-url>
```

This results in the DataSource being bound under the JNDI name "GenericDS" instead of the default of "java:/GenericDS" which restricts the lookup to the same VM as the jboss server.

> **Note**
>
> JBoss does not recommend using this feature on a production environment. It requires accessing a connection pool remotely and this is an anti-pattern as connections are not serializable. Besides, transaction propagation is not supported and it could lead to connection leaks if the remote clients are unreliable (i.e crashes, network failure). If you do need to access a datasource remotely, JBoss recommends accessing it via a remote session bean facade.

2.16. Configuring a DataSource to use login modules

Add the security-domain parameter to the *-ds.xml file.

```
<datasources>
<local-tx-datasource>
...
<security-domain>MyDomain</security-domain>
...
</local-tx-datasource>
</datasources>
```

Add an application-policy to the login-config.xml file. The authentication section should include the configuration for your login-module. For example, if you want to encrypt the database password, use the SecureIdentityLoginModule login module.

```
<application-policy name="MyDomain">
<authentication>
<login-module code="org.jboss.resource.security.SecureIdentityLoginModule" flag="required">
<module-option name="username">scott</module-option>
<module-option name="password">-170dd0fbd8c13748</module-option>
```

```
<module-option

module-option>
</login-module>
</authentication>
</application-policy>
```

In case you plan to fetch the data source connection from a web application, make sure authentication is turned on for the web application. This is in order for the Subject to be populated. If you wish for users to be able to connect anonymously, an additional login module needs to be added to the application-policy, in order to populate the security credentials. Add the UsersRolesLoginModule as the first login module in the chain. The usersProperties and rolesProperties parameters can be directed to dummy files.

```
<login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule" flag="required">
<module-option name="unauthenticatedIdentity">nobody</module-option>
<module-option name="usersProperties">props/users.properties</module-option>
<module-option name="rolesProperties">props/roles.properties</module-option>
</login-module>
```

Pooling

3.1. Strategy

JBossJCA [http://www.jboss.org/wiki/JBossJCA] uses a `ManagedConnectionPool` to perform the pooling. The `ManagedConnectionPool` is made up of subpools depending upon the strategy chosen and other pooling parameters.

xml	mbean	Internal Name	Description	
	ByNothing	OnePool	A single pool of equivalent connections	
<application-managed-security/>	ByApplication	PoolByCRI	Use the connection properties from allocateConnection()	
<security-domain/>	ByContainer	PoolBySubject	A pool per Subject, e.g. preconfigured or EJB/Web login subjects	
<security-domain-and-applicaton/>	ByContainerAndApplication	PoolBySubjectAndCri	A pool per Subject and connection property combination	

> **Note**
>
> The xml names imply this is just about security. This is misleading.

For <security-domain-and-application/> the Subject always overrides any user/password from createConnection(user, password) in the CRI:

```
(
ConnectionRequestInfo
)
```

3.2. Transaction stickness

You can force the same connection from a (sub-)pool to get reused throughout a transaction with the <track-connection-by-tx/> flag

> **Note**
>
> This is the only supported behaviour for *"local"* transactions. This element is deprecated in JBoss-5.x where transaction stickiness is enabled by default. XA users can explicitly enable interleaving with <interleaving/> element.

3.3. Workaround for Oracle

Oracle does not like XA connections getting used both inside and outside a JTA transaction. To workaround the problem you can create separate sub-pools for the different contexts using <no-tx-separate-pools/>.

3.4. Pool Access

The pool is designed for concurrent usage.

Upto <max-pool-size/> threads can be inside the pool at the same time (or using connections from a pool).

Once this limit is reached, threads wait for the <blocking-timeout-seconds/> to use the pool before throwing a *No Managed Connections Available* [http://www.jboss.org/wiki/WhatDoesTheMessageNoManagedConnectionsAvailableMean]

3.5. Pool Filling

The number of connections in the pool is controlled by the pool sizes.

- <min-pool-size/> - When the number of connections falls below this size, new connections are created
- <max-pool-size/> - No more than this number of connections are created
- <prefill/> - Feature Request has been implemented for 4.0.5. Note: the only pooling strategy that supports this feature is OnePool?, or ByNothing? pooling criteria.

The pool filling is done by a separate "Pool Filler" thread rather than blocking application threads.

3.6. Idle Connections

You can configure connections to be closed when they are idle. e.g. If you just had a peak period and now want to reap the unused ones. This is done via the <idle-timeout-minutes/>.

Idle checking is done on a separate "Idle Remover" thread on an LRU (least recently used) basis. The check is done every idle-timeout-minutes divided by 2 for connections unused for idle-timeout-minutes.

The pool itself operates on an MRU (most recently used) basis. This allows the excess connections to be easily identified.

Should closing idle connections cause the pool to fall below the min-pool-size, new/fresh connections are created.

> **Note**
>
> If you have long running transactions and you use interleaving (i.e. don't track-connection-by-tx) make sure the idle timeout is greater than the transaction timeout. When interleaving the connection is returned to the pool for others to use. If however nobody does use it, it would be a candidate for removal before the transaction is committed.

3.7. Dead connections

The JDBC protocol does not provide a natural `connectionErrorOccured()` event when a connection is broken. To support dead/broken connection checking there are a number of plugins.

3.7.1. Valid connection checking

The simplest format is to just run a "quick" sql statement:

```
<check-valid-connection-sql>select 1 from dual</check-valid-connection-sql>
```

before handing the connection to the application. If this fails, another connection is selected until there are no more connections at which point new connections are constructed.

The potentially more performant check is to use vendor specific features, e.g. Oracle's or MySQL's pingDatabase() via the

```
<valid-connection-checker-class-name/>
```

3.7.2. Errors during SQL queries

You can check if a connection broke during a query by the looking the error codes or messages of the SQLException for FATAL errors rather than normal SQLExceptions. These codes/messages can be vendor specific, e.g.

```
<exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter</
exception-sorter-class-name>
```

For

```
FATAL
```

errors the connection will be closed.

3.7.3. Changing/Closing/Flushing the pool

- *change or flush()* [http://www.jboss.org/wiki/HowDoIChangeThePoolingParameters] the pool
- closing/undeploying the pool will do a flush first

3.7.4. Other pooling

Thirdparty Pools [http://www.jboss.org/wiki/IWantToPluginACustomThirdpartyDataSource] - only if you know what you are doing

Part III. Reference

Connectors on JBoss

The JCA Configuration and Architecture

This chapter discusses the JBoss server implementation of the JavaEE Connector Architecture (JCA). JCA is a resource manager integration API whose goal is to standardize access to non-relational resources in the same way the JDBC API standardized access to relational data. The purpose of this chapter is to introduce the utility of the JCA APIs and then describe the architecture of JCA in JBoss

4.1. An Overview of the JBoss JCA Architecture

The JBoss JCA framework provides the application server architecture extension required for the use of JCA resource adaptors. This is primarily a connection pooling and management extension along with a number of MBeans for loading resource adaptors into the JBoss server.

There are three coupled MBeans that make up a RAR deployment. These are the `org.jboss.resource.deployment.RARDeployment`, `org.jboss.resource.connectionmanager.RARDeployment`, and `org.jboss.resource.connectionmanager.BaseConnectionManager2`. The `org.jboss.resource.deployment.RARDeployment` is simply an encapsulation of the metadata of a RAR `META-INF/ra.xml` descriptor. It exposes this information as a DynamicMBean simply to make it available to the `org.jboss.resource.connectionmanager.RARDeployment` MBean.

The RARDeployer service handles the deployment of archives files containing resource adaptors (RARs). It creates the `org.jboss.resource.deployment.RARDeployment` MBeans when a RAR file is deployed. Deploying the RAR file is the first step in making the resource adaptor available to application components. For each deployed RAR, one or more connection factories must be configured and bound into JNDI. This task performed using a JBoss service descriptor that sets up a `org.jboss.resource.connectionmanager.BaseConnectionManager2` MBean implementation with a `org.jboss.resource.connectionmgr.RARDeployment` dependent.

4.1.1. BaseConnectionManager2 MBean

The `org.jboss.resource.connectionmanager.BaseConnectionManager2` MBean is a base class for the various types of connection managers required by the JCA spec. Subclasses include `NoTxConnectionManager`, `LocalTxConnectionManager` and `XATxConnectionManager`. These correspond to resource adaptors that support no transactions, local transaction and XA transaction respectively. You choose which subclass to use based on the type of transaction semantics you want, provided the JCA resource adaptor supports the corresponding transaction capability.

The common attributes supported by the BaseConnectionManager2 MBean are:

- **ManagedConnectionPool**: This specifies the ObjectName of the MBean representing the pool for this connection manager. The MBean must

have an `ManagedConnectionPool` attribute that is an implementation of the `org.jboss.resource.connectionmanager.ManagedConnectionPool` interface. Normally it will be an embedded MBean in a depends tag rather than an `ObjectName` reference to an existing MBean. The default MBean for use is the `org.jboss.resource.connectionmanager.JBossManagedConnectionPool`. Its configurable attributes are discussed below.

- **CachedConnectionManager**: This specifies the `ObjectName` of the `CachedConnectionManager` MBean implementation used by the connection manager. Normally this is specified using a depends tag with the `ObjectName` of the unique `CachedConnectionManager` for the server. The name `jboss.jca:service=CachedConnectionManager` is the standard setting to use.

- **SecurityDomainJndiName**: This specifies the JNDI name of the security domain to use for authentication and authorization of resource connections. This is typically of the form `java:/jaas/<domain>` where the `<domain>` value is the name of an entry in the `conf/login-config.xml` JAAS login module configuration file. This defines which JAAS login modules execute to perform authentication.

- **JaasSecurityManagerService**: This is the `ObjectName` of the security manager service. This should be set to the security manager MBean name as defined in the `conf/jboss-service.xml` descriptor, and currently this is `jboss.security:service=JaasSecurityManager`. This attribute will likely be removed in the future.

4.1.2. RARDeployment MBean

The `org.jboss.resource.connectionmanager.RARDeployment` MBean manages configuration and instantiation `ManagedConnectionFactory` instance. It does this using the resource adaptor metadata settings from the RAR `META-INF/ra.xml` descriptor along with the `RARDeployment` attributes. The configurable attributes are:

- **OldRarDeployment**: This is the `ObjectName` of the `org.jboss.resource.RarDeployment` MBean that contains the resource adaptor metadata. The form of this name is `jboss.jca:service=RARDeployment,name=<ra-display-name>` where the `<ra-display-name>` is the `ra.xml` descriptor `display-name` attribute value. The `RARDeployer` creates this when it deploys a RAR file. This attribute will likely be removed in the future.

- **ManagedConnectionFactoryProperties**: This is a collection of (name, type, value) triples that define attributes of the `ManagedConnectionFactory` instance. Therefore, the names of the attributes depend on the resource adaptor `ManagedConnectionFactory` instance. The following example shows the structure of the content of this attribute.

```
<properties>
   <config-property>
      <config-property-name>Attr0Name</config-property-name>
```

```
      <config-property-type>Attr0Type</config-property-type>
      <config-property-value>Attr0Value</config-property-value>
   </config-property>
   <config-property>
      <config-property-name>Attr1Name</config-property-name>
      <config-property-type>Attr2Type</config-property-type>
      <config-property-value>Attr2Value</config-property-value>
   </config-property>
   ...
</properties>
```

`AttrXName` is the Xth attribute name, `AttrXType` is the fully qualified Java type of the attribute, and `AttrXValue` is the string representation of the value. The conversion from string to `AttrXType` is done using the `java.beans.PropertyEditor` class for the `AttrXType`.

- **JndiName**: This is the JNDI name under which the resource adaptor will be made available. Clients of the resource adaptor use this name to obtain either the `javax.resource.cci.ConnectionFactory` or resource adaptor specific connection factory. The full JNDI name will be `java:/<JndiName>` meaning that the `JndiName` attribute value will be prefixed with `java:/`. This prevents use of the connection factory outside of the JBoss server VM. In the future this restriction may be configurable.

4.1.3. JBossManagedConnectionPool MBean

The `org.jboss.resource.connectionmanager.JBossManagedConnectionPool` MBean is a connection pooling MBean. It is typically used as the embedded MBean value of the `BaseConnectionManager2ManagedConnectionPool` attribute. When you setup a connection manager MBean you typically embed the pool configuration in the connection manager descriptor. The configurable attributes of the `JBossManagedConnectionPool` are:

- **ManagedConnectionFactoryName**: This specifies the `ObjectName` of the MBean that creates `javax.resource.spi.ManagedConnectionFactory` instances. Normally this is configured as an embedded MBean in a depends element rather than a separate MBean reference using the `RARDeployment` MBean. The MBean must provide an appropriate `startManagedConnectionFactory` operation.
- **MinSize**: This attribute indicates the minimum number of connections this pool should hold. These are not created until a `Subject` is known from a request for a connection. `MinSize` connections will be created for each sub-pool.
- **MaxSize**: This attribute indicates the maximum number of connections for a pool. No more than MaxSize connections will be created in each sub-pool.
- **BlockingTimeoutMillis**: This attribute indicates the maximum time to block while waiting for a connection before throwing an exception. Note that this blocks only while waiting for a permit

for a connection, and will never throw an exception if creating a new connection takes an inordinately long time.

- **IdleTimeoutMinutes**: This attribute indicates the maximum time a connection may be idle before being closed. The actual maximum time depends also on the idle remover thread scan time, which is 1/2 the smallest idle timeout of any pool.

- **NoTxSeparatePools**: Setting this to true doubles the available pools. One pool is for connections used outside a transaction the other inside a transaction. The actual pools are lazily constructed on first use. This is only relevant when setting the pool parameters associated with the `LocalTxConnectionManager` and `XATxConnectionManager`. Its use case is for Oracle (and possibly other vendors) XA implementations that don't like using an XA connection with and without a JTA transaction.

- **Criteria**: This attribute indicates if the JAAS `javax.security.auth.Subject` from security domain associated with the connection, or app supplied parameters (such as from `getConnection(user, pw)`) are used to distinguish connections in the pool. The allowed values are:
  - **ByContainer**: use `Subject`
  - **ByApplication**: use application supplied parameters only
  - **ByContainerAndApplication**: use both
  - **ByNothing**: all connections are equivalent, usually if adapter supports reauthentication

4.1.4. CachedConnectionManager MBean

The `org.jboss.resource.connectionmanager.CachedConnectionManager` MBean manages associations between meta-aware objects (those accessed through interceptor chains) and connection handles, as well as between user transactions and connection handles. Normally there should only be one such MBean, and this is configured in the core `jboss-service.xml` descriptor. It is used by `CachedConnectionInterceptor`, JTA `UserTransaction` implementation and all `BaseConnectionManager2` instances. The configurable attributes of the `CachedConnectionManager` MBean are:

- **SpecCompliant**: Enable this boolean attribute for spec compliant non-shareable connections reconnect processing. This allows a connection to be opened in one call and used in another. Note that specifying this behavior disables connection close processing.

- **Debug**: Enable this boolean property for connection close processing. At the completion of an EJB method invocation, unclosed connections are registered with a transaction synchronization. If the transaction ends without the connection being closed, an error is reported and JBoss closes the connection. This is a development feature that should be turned off in production for optimal performance.

- **TransactionManagerServiceName**: This attribute specifies the JMX `ObjectName` of the JTA transaction manager service. Connection close processing is now synchronized with the transaction manager and this attribute specifies the transaction manager to use.

4.1.5. A Sample Skeleton JCA Resource Adaptor

To conclude our discussion of the JBoss JCA framework we will create and deploy a single non-transacted resource adaptor that simply provides a skeleton implementation that stubs out the required interfaces and logs all method calls. We will not discuss the details of the requirements of a resource adaptor provider as these are discussed in detail in the JCA specification. The purpose of the adaptor is to demonstrate the steps required to create and deploy a RAR in JBoss, and to see how JBoss interacts with the adaptor.

The adaptor we will create could be used as the starting point for a non-transacted file system adaptor. The source to the example adaptor can be found in the `src/main/org/jboss/book/jca/ex1` directory of the book examples. A class diagram that shows the mapping from the required `javax.resource.spi` interfaces to the resource adaptor implementation is given in *Figure 4.1, "The file system RAR class diagram"*.

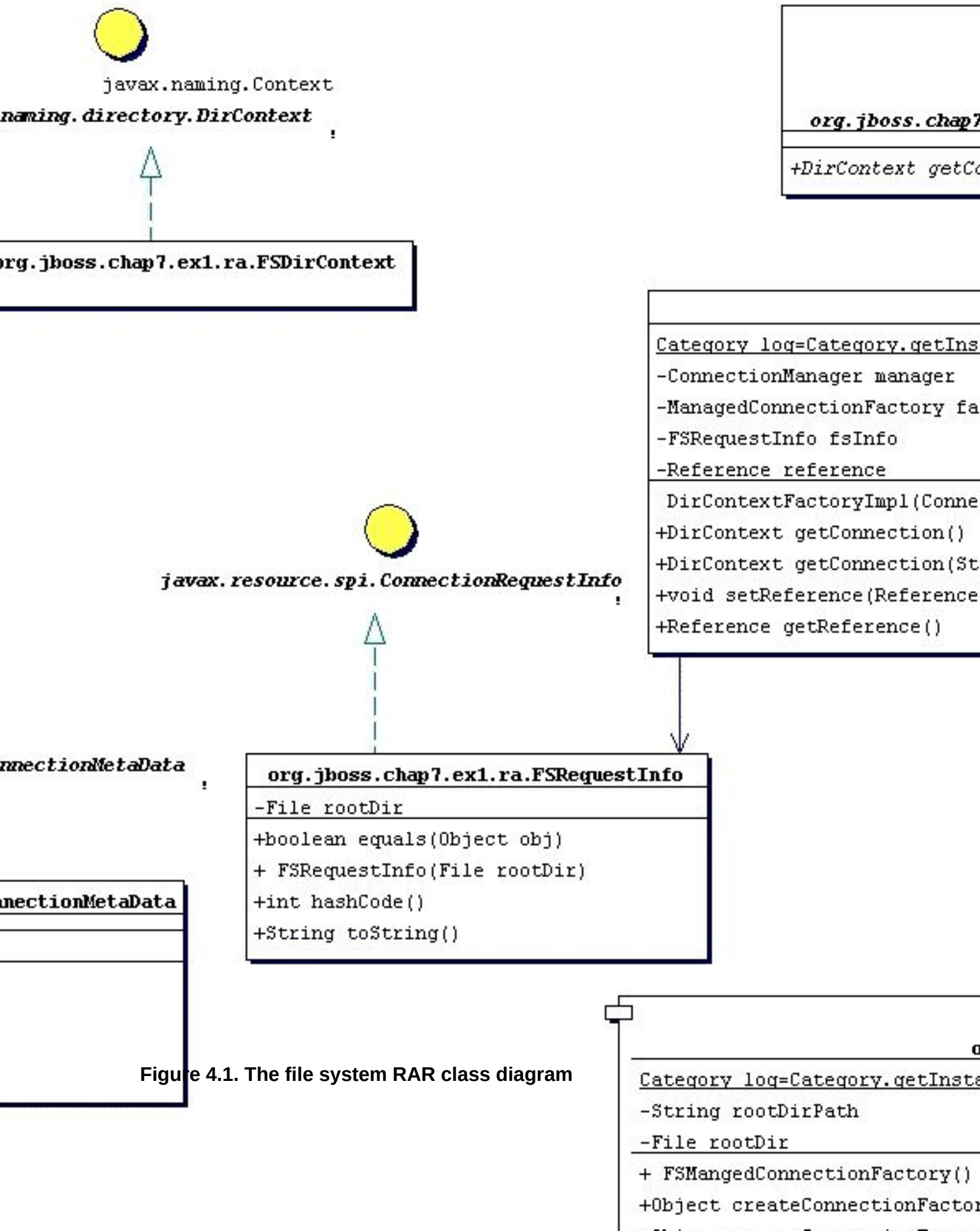

Figure 4.1. The file system RAR class diagram

We will build the adaptor, deploy it to the JBoss server and then run an example client against an EJB that uses the resource adaptor to demonstrate the basic steps in a complete context. We'll then take a look at the JBoss server log to see how the JBoss JCA framework interacts with the resource adaptor to help you better understand the components in the JCA system level contract.

To build the example and deploy the RAR to the JBoss server `deploy/lib` directory, execute the following Ant command in the book examples directory.

```
[examples]$ ant -Dchap=jca build-chap
```

The deployed files include a `jca-ex1.sar` and a `notxfs-service.xml` service descriptor. The example resource adaptor deployment descriptor is shown in *Example 4.1, "The nontransactional file system resource adaptor deployment descriptor."*.

Example 4.1. The nontransactional file system resource adaptor deployment descriptor.

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://java.sun.com/xml/ns/connector
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                  http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd" version="1.5">
   <display-name>File System Adapter</display-name>
   <vendor-name>JBoss</vendor-name>
   <eis-type>FileSystem</eis-type>
   <resourceadapter-version>1.0</resourceadapter-version>
   <license>
      <description>LGPL</description>
      <license-required>false</license-required>
   </license>
   <resourceadapter>
      <resourceadapter-class>
         org.jboss.resource.deployment.DummyResourceAdapter
      </resourceadapter-class>
      <outbound-resourceadapter>
         <connection-definition>
                                                <managedconnectionfactory-class>
  org.jboss.book.jca.ex1.ra.FSManagedConnectionFactory  </managedconnectionfactory-
class>
            <config-property>
               <config-property-name>FileSystemRootDir</config-property-name>
               <config-property-type>java.lang.String</config-property-type>
               <config-property-value>/tmp/db/fs_store</config-property-value>
```

```
            </config-property>
            <config-property>
               <config-property-name>UserName</config-property-name>
               <config-property-type>java.lang.String</config-property-type>
               <config-property-value/>
            </config-property>
            <config-property>
               <config-property-name>Password</config-property-name>
               <config-property-type>java.lang.String</config-property-type>
               <config-property-value/>
            </config-property>
                                                                    <connectionfactory-
interface> org.jboss.book.jca.ex1.ra.DirContextFactory </connectionfactory-interface>
    <connectionfactory-impl-class> org.jboss.book.jca.ex1.ra.DirContextFactoryImpl </
connectionfactory-impl-class> <connection-interface> javax.naming.directory.DirContext
 </connection-interface> <connection-impl-class> org.jboss.book.jca.ex1.ra.FSDirContext
 </connection-impl-class>
         </connection-definition>
         <transaction-support>NoTransaction</transaction-support>
         <authentication-mechanism>
            <authentication-mechanism-type>BasicPassword</authentication-mechanism-type>
            <credential-interface>
               javax.resource.spi.security.PasswordCredential
            </credential-interface>
         </authentication-mechanism>
         <reauthentication-support>true</reauthentication-support>
      </outbound-resourceadapter>
      <security-permission>
         <description> Read/Write access is required to the contents of the
            FileSystemRootDir </description>
         <security-permission-spec> permission java.io.FilePermission
            "/tmp/db/fs_store/*", "read,write";
         </security-permission-spec>
      </security-permission>
   </resourceadapter>
</connector>
```

The key items in the resource adaptor deployment descriptor are highlighted in bold. These define the classes of the resource adaptor, and the elements are:

- **managedconnectionfactory-class**: The implementation of the `ManagedConnectionFactory` interface, `org.jboss.book.jca.ex1.ra.FSManagedConnectionFactory`

- **connectionfactory-interface**: This is the interface that clients will obtain when they lookup the connection factory instance from JNDI, here a proprietary resource adaptor value, `org.jboss.book.jca.ex1.ra.DirContextFactory`. This value will be needed when we create the JBoss `ds.xml` to use the resource.
- **connectionfactory-impl-class**: This is the class that provides the implementation of the `connectionfactory-interface`, `org.jboss.book.jca.ex1.ra.DirContextFactoryImpl`.
- **connection-interface**: This is the interface for the connections returned by the resource adaptor connection factory, here the JNDI `javax.naming.directory.DirContext` interface.
- **connection-impl-class**: This is he class that provides the `connection-interface` implementation, `org.jboss.book.jca.ex1.ra.FSDirContext`.
- **transaction-support**: The level of transaction support, here defined as `NoTransaction`, meaning the file system resource adaptor does not do transactional work.

The RAR classes and deployment descriptor only define a resource adaptor. To use the resource adaptor it must be integrated into the JBoss application server using a `ds.xml` descriptor file. An example of this for the file system adaptor is shown in *Example 4.2, "The notxfs-ds.xml resource adaptor MBeans service descriptor."*.

Example 4.2. The notxfs-ds.xml resource adaptor MBeans service descriptor.

```
<!DOCTYPE connection-factories PUBLIC
        "-//JBoss//DTD JBOSS JCA Config 1.5//EN"
        "http://www.jboss.org/j2ee/dtd/jboss-ds_1_5.dtd">
<!--
      The non-transaction FileSystem resource adaptor service configuration
-->
<connection-factories>
   <no-tx-connection-factory>
      <jndi-name>NoTransFS</jndi-name>
      <rar-name>jca-ex1.rar</rar-name>
      <connection-definition>
         org.jboss.book.jca.ex1.ra.DirContextFactory
      </connection-definition>
      <config-property name="FileSystemRootDir"
                  type="java.lang.String">/tmp/db/fs_store</config-property>
   </no-tx-connection-factory>
</connection-factories>
```

The main attributes are:

- **jndi-name**: This specifies where the connection factory will be bound into JNDI. For this deployment that binding will be `java:/NoTransFS`.
- **rar-name**: This is the name of the RAR file that contains the definition for the resource we want to provide. For nested RAR files, the name would look like `myapplication.ear#my.rar`. In this example, it is simply `jca-ex1.rar`.
- **connection-definition**: This is the connection factory interface class. It should match the `connectionfactory-interface` in the `ra.xml` file. Here our connection factory interface is `org.jboss.book.jca.ex1.ra.DirContextFactory`.
- **config-property**: This can be used to provide non-default settings to the resource adaptor connection factory. Here the `FileSystemRootDir` is being set to `/tmp/db/fs_store`. This overrides the default value in the `ra.xml` file.

To deploy the RAR and connection manager configuration to the JBoss server, run the following:

```
[examples]$ ant -Dchap=jca config
```

The server console will display some logging output indicating that the resource adaptor has been deployed.

Now we want to test access of the resource adaptor by a JavaEE component. To do this we have created a trivial stateless session bean that has a single method called `echo`. Inside of the `echo` method the EJB accesses the resource adaptor connection factory, creates a connection, and then immediately closes the connection. The `echo` method code is shown below.

Example 4.3. The stateless session bean echo method code that shows the access of the resource adaptor connection factory.

```
public String echo(String arg)
{
   log.info("echo, arg="+arg);
   try {
      InitialContext ctx = new InitialContext();
      Object         ref = ctx.lookup("java:comp/env/ra/DirContextFactory");
      log.info("echo, ra/DirContextFactory=" + ref);

      DirContextFactory dcf = (DirContextFactory) ref;
      log.info("echo, found dcf=" + dcf);

      DirContext dc = dcf.getConnection();
      log.info("echo, lookup dc=" + dc);
```

```
        dc.close();
    } catch(NamingException e) {
        log.error("Failed during JNDI access", e);
    }
    return arg;
}
```

The EJB is not using the CCI interface to access the resource adaptor. Rather, it is using the resource adaptor specific API based on the proprietary `DirContextFactory` interface that returns a JNDI `DirContext` object as the connection object. The example EJB is simply exercising the system contract layer by looking up the resource adaptor connection factory, creating a connection to the resource and closing the connection. The EJB does not actually do anything with the connection, as this would only exercise the resource adaptor implementation since this is a non-transactional resource.

Run the test client which calls the `EchoBean.echo` method by running Ant as follows from the examples directory:

```
[examples]$ ant -Dchap=jca -Dex=1 run-example
```

You'll see some output from the bean in the system console, but much more detailed logging output can be found in the `server/production/log/server.log` file. Don't worry if you see exceptions. They are just stack traces to highlight the call path into parts of the adaptor. To help understand the interaction between the adaptor and the JBoss JCA layer, we'll summarize the events seen in the log using a sequence diagram. *Figure 4.2, “A sequence diagram illustrating the key interactions between the JBoss JCA framework and the example resource adaptor that result when the EchoBean accesses the resource adaptor connection factory.”* is a sequence diagram that summarizes the events that occur when the `EchoBean` accesses the resource adaptor connection factory from JNDI and creates a connection.

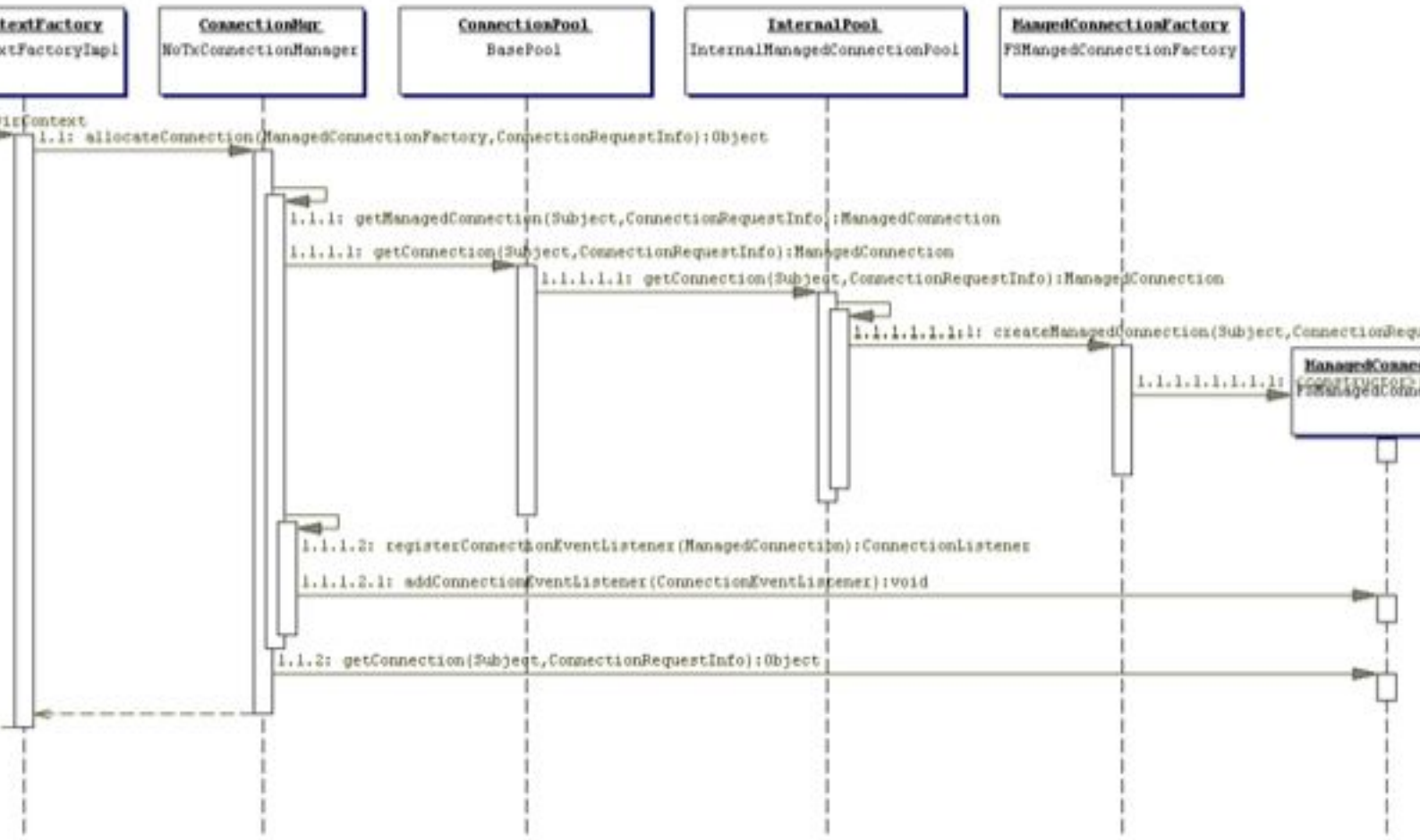

Figure 4.2. A sequence diagram illustrating the key interactions between the JBoss JCA framework and the example resource adaptor that result when the EchoBean accesses the resource adaptor connection factory.

The starting point is the client's invocation of the `EchoBean.echo` method. For the sake of conciseness of the diagram, the client is shown directly invoking the EchoBean.echo method when in reality the JBoss EJB container handles the invocation. There are three distinct interactions between the `EchoBean` and the resource adaptor; the lookup of the connection factory, the creation of a connection, and the close of the connection.

The lookup of the resource adaptor connection factory is illustrated by the 1.1 sequences of events. The events are:

- the echo method invokes the `getConnection` method on the resource adaptor connection factory obtained from the JNDI lookup on the `java:comp/env/ra/DirContextFactory` name which is a link to the `java:/NoTransFS` location.

- the `DirContextFactoryImpl` class asks its associated `ConnectionManager` to allocate a connection. It passes in the `ManagedConnectionFactory` and `FSRequestInfo` that were associated with the `DirContextFactoryImpl` during its construction.
- the `ConnectionManager` invokes its `getManagedConnection` method with the current `Subject` and `FSRequestInfo`.
- the `ConnectionManager` asks its object pool for a connection object. The `JBossManagedConnectionPool$BasePool` is get the key for the connection and then asks the matching `InternalPool` for a connection.
- Since no connections have been created the pool must create a new connection. This is done by requesting a new managed connection from the `ManagedConnectionFactory`. The `Subject` associated with the pool as well as the `FSRequestInfo` data are passed as arguments to the `createManagedConnection` method invocation.
- the `ConnectionFactory` creates a new `FSManagedConnection` instance and passes in the `Subject` and `FSRequestInfo` data.
- a `javax.resource.spi.ConnectionListener` instance is created. The type of listener created is based on the type of `ConnectionManager`. In this case it is an `org.jboss.resource.connectionmgr.BaseConnectionManager2$NoTransactionListener` instance.
- the listener registers as a `javax.resource.spi.ConnectionEventListener` with the `ManagedConnection` instance created in 1.2.1.1.
- the `ManagedConnection` is asked for the underlying resource manager connection. The `Subject` and `FSRequestInfo` data are passed as arguments to the `getConnection` method invocation.
- The resulting connection object is cast to a `javax.naming.directory.DirContext` instance since this is the public interface defined by the resource adaptor.
- After the `EchoBean` has obtained the `DirContext` for the resource adaptor, it simply closes the connection to indicate its interaction with the resource manager is complete.

This concludes the resource adaptor example. Our investigation into the interaction between the JBoss JCA layer and a trivial resource adaptor should give you sufficient understanding of the steps required to configure any resource adaptor. The example adaptor can also serve as a starting point for the creation of your own custom resource adaptors if you need to integrate non-JDBC resources into the JBoss server environment.

4.2. XA Recovery in the JCA layer

Describes the design of how the JCA layer registers XA datasource for XA Resource Recovery with the JBoss TS project.

4.2.1. New -ds.xml file

We will create a jboss-ds_5_1.dtd (EAP 5.1) / jboss-ds_6_0.dtd (AS 6) which has the additional fields of

- recover-user-name
- recover-password
- recover-security-domain
- no-recover

The first two will represent a user and password pair which has the credentials to perform the recovery operation. The third likewise, but using a security domain instead. The last field is to exclude a datasource from recovery.

The fields should have a fall back value of their non-recover counterparts - e.g. user-name, password and security-domain.

4.2.2. Minimal changes

We should limit the main changes to
`org.jboss.resource.connectionmanager.ManagedConnectionFactoryDeployment`

which controls the ManagedConnectionFactory for the resource adapter.

4.2.3. Availability

- JBoss Enterprise Application Platform 5.1 or higher
- JBoss Application Server 6.0.0.M4 or higher