Reladomo Middle Tier

Table of Contents

1. Reladomo Middle Tier

1.1. Why?

- Security (fat client applications):
    - User ID must not be able to access database directly (especially write)
    - Batch/App ID must not be used from unauthorized IP’s
    - For a large, semi-mobile user community, maintaining IP lists is undesirable and allows random sql client connections
- Connection sharing: database connections can be expensive. Many users can share same connection.

1.2. How?

- Third tier acts like a relational source. Supports relational-like operations: find, insert, update, delete.
- No object graphs. Not a complex object source. Serializaton based on metadata. Wire format looks like a result set.
- Lightweight: can be configured as pass-through with no caching.
- Remoting API must be implemented by application.

2. Setup

You'll need to setup a server that loads a normal Reladomo configuration. Additionally, the server has to expose a remote interface for its clients to talk to. There is a PSP implementation for that remote service. It's often convenient to use the PSP servlet's load initialization to initialize the underlying Reladomo implementation.

2.1. Servlet

This is an example setup where we have a war file that has web.xml containing the following snippet:

Example 1. web.xml fragment

```
<servlet>
    <servlet-name>PspServlet</servlet-name>
    <servlet-class>com.gs.fw.common.base.psp.server.PspServlet</
servlet-class>
    <init-param>
        <param-name>serviceInterface.MyappRemoteMithraService</param-
name>
        <param-
value>com.gs.fw.common.mithra.remote.RemoteMithraService</param-value>
    </init-param>
    <init-param>
        <param-name>serviceClass.MyAppRemoteMithraService</param-name>
        <param-value>com.gs.fw.para.common.MyAppRemoteMithraService</
param-value>
    </init-param>
    <load-on-startup>10</load-on-startup>
</servlet>
```

2.2. Remote Service

The constructor of MyAppRemoteMithraService is essentially used to initialize Reladomo.

Example 2. Sample implementation of a RemoteMithraService

```
public class MyAppRemoteMithraService extends RemoteMithraServiceImpl
{
    private static final Logger logger =
 LoggerFactory.createLogger(MyAppRemoteMithraService.class);

    public MyAppRemoteMithraService()
    {
        String remoteMithraApplicationName =
 System.getProperty("mithra.middle.tier.loader");
        if (remoteMithraApplicationName == null)
        {
            remoteMithraApplicationName
 = "MyAppMithraMiddleTierLoader";
        }
        try
        {

 ApplicationRunner.instance().runApplication(remoteMithraApplicationName);
            logger.info("Successfully loaded Reladomo middle tier.");
        }
        catch (Exception e)
        {
            logger.error("Could not load Reladomo middle tier", e);
            throw new MithraBusinessException("could not load Reladomo
 middle tier", e);
```

```
        }
    }
}
```

The actual remote implementation comes from RemoteMithraServiceImpl. The Reladomo runtime configuration for the server will have a real connection manger.

2.3. Client-Side Runtime Configuration

On the client side, you'll have to change your runtime config to point to the middle tier:

Example 3. Sample client runtime configuration fragment

```
<MithraRuntime>

  <RemoteServer className="com.gs.fw.common.mithra.remote.PspBasedMithraRemoteServ
        <Property name="url" value="http://localhost:7399/
mithramiddletier/PspServlet"/>


  <MithraObjectConfiguration className="com.gs.fw.common.mithra.test.domain.Accoun
>

  <MithraObjectConfiguration className="com.gs.fw.common.mithra.test.domain.Specia
>
    </RemoteServer>
</MithraRuntime>
```

In a client runtime configuration, you can think that RemoteServer replaces ConnectionManager. The MithraObjectConfigurations here are completely optional, and simply overrides the values that come from the server. The client will setup a partialCache for the classes that the server is serving but not listed.

2.4. PspBasedMithraRemoteServerFactory

Here is an example, which receives the properties configured in the client runtime config XML, and returns configured instances:

Example 4. PspBasedMithraRemoteServerFactory

```
public class PspBasedMithraRemoteServerFactory
{
    public static RemoteMithraService getInstance(Properties
 properties)
    {
        String url = properties.getProperty("url");
        if (url == null)
        {
            throw new MithraException("you must specify a url property
 for this factory");
```

```
        }
        return createFactory(properties, url);
    }

    protected static RemoteMithraService createFactory(Properties
 properties, String url)
    {
        FastServletProxyFactory factory;
        String user = properties.getProperty("user");
        if (user != null)
        {
            String password = properties.getProperty("password");
            if (password == null)
            {
                throw new MithraException("if you specify the 'user'
 property, you must also specify a password!");
            }

            factory = new FastServletProxyFactory(user, password);
        }
        else
        {
            factory = new FastServletProxyFactory();
        }

        try
        {
            return factory.create(RemoteMithraService.class, url);
        }
        catch (MalformedURLException e)
        {
            throw new MithraException("malformed url: "+url, e);
        }
    }
}
```

The `RemoteMithraService` returned by the `createFactory()` method will manage the transfer of requests and results between the client and the middle tier.

Note:

`PspBasedMithraRemoteServerFactory` is available from Reladomo for your use.