Reladomo Notification

Table of Contents

1. What is "Reladomo Notification" and when do I need it?

When you have multiple JVMs connecting to a DB via Reladomo, you need to keep each JVM up-to-date with changes made by any of the other JVMs. Reladomo Notification is the primary mechanism for achieving this and keeping each JVMs Reladomo cache fresh.

If you only have one JVM accessing a DB, you don’t need Notification.

You will need Notification in any of the following situations:

- Two or more JVMs accessing a single DB
- An app server, plus a separate "batch updater" process
- You have hot-hot or hot-warm redundant processes

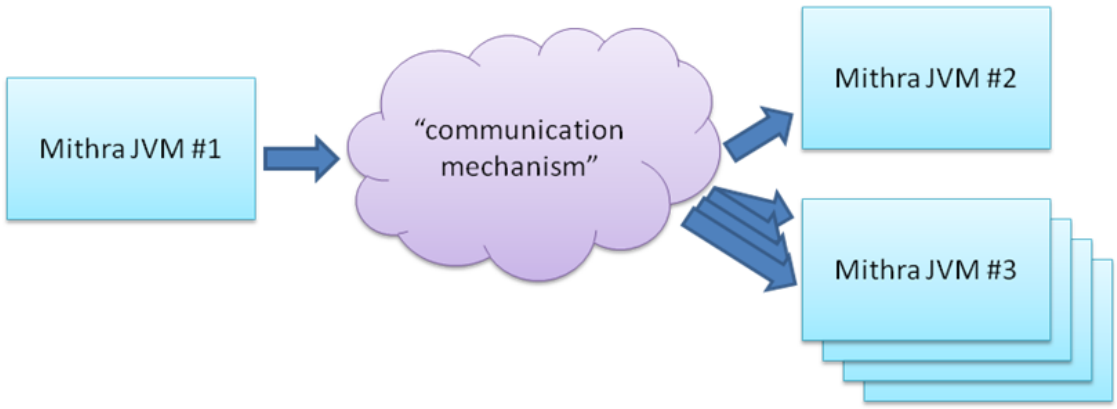
2. When does Notification not help?

Notification doesn't help when one or more of your processes are not Reladomo JVM based, since the notifications are generated by Reladomo itself. There are strategies for managing this situation, but are less granular than what Reladomo offers, and you should consult your local Reladomo expert before implementing.

3. How does Reladomo Notification work?

When DB data is changed by a Reladomo process, "notifications" of these changes are broadcast via some communication mechanism to all other Reladomo clients in your notification "realm".

Figure 1. Notification Communication



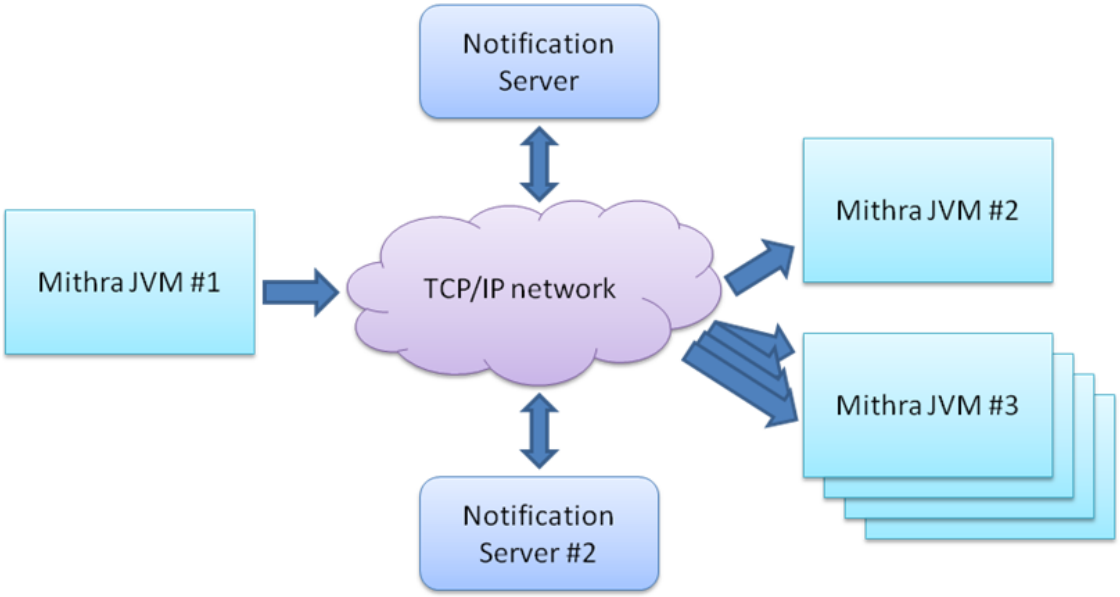
4. What "communication mechanisms" are available?

The easiest to use (and is appropriate in 99.9% of use-cases) is TCP/IP. There also exists an RV implementation or you could implement your own mechanism (neither are recommended though, and not discussed in this document).

5. What does the TCP-based Notification service involve?

Using classes provided with Reladomo, you can perform some simple configuration to enable Notification.

Figure 2. TCP Communication



The Notification Server is the central point that all JVMs will register with, and is responsible for forwarding notifications.

Note:

You will only have a second Notification server if you are using the "dual" communication mechanism.

A Notification server requires only a small amount of memory and CPU to execute, and is highly reliable - the biggest threats to a Notification server are: admin error or a machine failure (including unplanned power-down). To protect against one of these, we recommend using "keep-alive" to monitor the process and re-start it if it should ever go down in your Prod environment.

There are two varieties of TCP notifications: single notification and dual notification, and both leverage TCP/IP sockets. In practice, TCP communication is occasionally unreliable for brief periods, so Reladomo TCP notification supports re-tries to help ensure no data is lost:

- If a JVM loses its connection to a Notification server, it will periodically try to re-establish until it a connection is successfully made.
- If a Notification server loses a connection to a client, it will try to re-connect for a period, and if unsuccessful, will drop the client from its broadcast list. A client can re-join a server at any time, even if it was dropped from its broadcast list.

There are two varieties of TCP-based communication mechanisms: single notification (good for most use-cases) and dual notification (used when extreme redundancy and a low tolerance for stale data is demanded), both are discussed below.

In either situation, you will need to ensure your ConnectionManager is configured to return an appropriate DB identifier.

Example 1. Reladomo ConnectionManager getDatabaseIdentifier example implementation

```
public String getDatabaseIdentifier() {
    return serverName + ":" + schema;  // must not return null
}
```

5.1. Single Notification

This type leverages a single Notification server, and is very easy to configure in each of your JVM processes. At some point in your applications Reladomo instance initialization, add the following lines:

Example 2. Reladomo single TCP initialization fragment

```
MithraManagerProvider.getMithraManager().setNotificationEventManager(
        new MithraNotificationEventManagerImpl(
                new TcpMessagingAdapterFactory(host, port)));
```

5.2. Dual Notification

This type leverages two Notification servers, and is very easy to configure in each of your JVM processes. At some point in your applications Reladomo instance initialization, add the following lines:

Example 3. Reladomo dual TCP initialization fragment

```
MithraManagerProvider.getMithraManager().setNotificationEventManager(
        new MithraNotificationEventManagerImpl(
                new TcpDualMessagingAdapterFactory(host1, port1,
 host2, port2)));
```

With this configuration, notification messages are duplicated using redundant communication clients to redundant Notification servers.

5.3. Starting a Notification Server

Reladomo ships with all the necessary libraries to launch a Notification server, so starting a new instance is very easy:

Example 4. Command line to launch a Notification Server

```
java -classpath <all the required jars> -Dport=<port number>
        com.gs.fw.common.mithra.notification.server.NotificationServer
```

If you are using dual-notification, you will need to do this twice, preferably on two different physical machines (for better robustness).

6. Anything else I should consider?

Yes.

6.1. cacheTimeToLive

We strongly recommend setting a `cacheTimeToLive` value of 5 minutes (a.k.a. 300,000 ms) in your Reladomo runtime-XML configurations for all the `partialCache` types. E.g.:

Example 5. Reladomo Runtime XML file cache configuration fragment

```
<MithraObjectConfiguration className="com.gs.dept.app.domain.Order" cacheType="par
 />
```

Should connectivity be lost to a Notification server, but the DB connection is still good, JVM processes can continue to work, but are at a small risk of having stale data, hence a time-out for cached data helps maintain data freshness whilst also helping to minimize peak memory usage.

6.2. Data integrity

If connectivity to a Notification server is lost, but the DB connection is still good, JVM processes can continue to work, but are at a small risk of having stale data. You can minimize this by setting the `cacheTimeToLive` value as shown above. Also note that any transactional data being updated will be protected due to a combination of the transaction enrolling data rows being modified and optimistic locking.