# HA Joram user guide

# 1 Platform Configuration

A HA Joram server is launched on a cluster of ScalAgent servers. So the configuration of a HA Joram server implies to configure a clustered ScalAgent server.

## 1.1 Clustered ScalAgent server configuration

A clustered ScalAgent server is defined by the element “cluster”. A cluster owns an identifier and a name defined by the attributes “id” and “name” (exactly like a standard server). Two properties must be defined:

- “Engine” must be set to “fr.dyade.aaa.agent.HATransactionEngine” which is the class name of the engine that provides high availability (*set by default in a next version* ?).
- “nbClusterExpected” defines the number of replicas that must be connected to the group communication channel used before this replica starts

```
<?xml version="1.0"?>
<config>
  <domain name="D1"/>
  <cluster id="0" name="s0">
    <property name="Engine"
              value="fr.dyade.aaa.agent.HATransactionEngine"/>
    <property name="nbClusterExpected" value="3"/>
```

For each replica an element “server” must be added. The attribute “id” defines the identifier of the replica inside the cluster. The attribute “hostname” gives the address of the host where the replica is running. A server replica is defined exactly like a standard server. For example this one owns a network and two services. The network is used by the replica to communicate with external agent servers, i.e. servers located outside of the cluster and not replicas.

```
    <server id="0" hostname="localhost">
       <network domain="D1" port="16301"/>
    </server>
```

Two other replicas are defined in the clustered ScalAgent server s0. Notice that they define their network with a different port value because they work on the same host (localhost).

```
    <server id="1" hostname="localhost">
      <network domain="D1" port="16302"/>
    </server>
    <server id="2" hostname="localhost">
      <network domain="D1" port="16303"/>
    </server>
  </cluster>
</config>
```

## 1.2 Group communication

There are two properties to configure the address and the port of the IP multicast used by JGroups: *JGroups.MCastAddr* and *JGroups.McastPort*.

The group communication semantic is not configurable as it requires to understand exactly how the high availability works. However some higher level configuration handles will be given if needed.

## 1.3 Joram server configuration

A Joram server is started by a ScalAgent service called "ConnectionManager" (see Joram documentation).

```
<service class="org.objectweb.joram.mom.proxies.ConnectionManager"
         args="root root"/>
```

This service must be added to all the replicas. Some more services can be added to each replicas according to the entry points needed by the clients: Tcp, Soap and collocated.

### 1.3.1 TCP entry point

The service "TcpProxyService" starts a TCP entry point to the Joram server.

```
<service class="org.objectweb.joram.mom.proxies.
                tcp.TcpProxyService"
         args="2560"/>
```

### 1.3.2 Soap entry point

*Not yet available (in HA mode)*

### 1.3.3 Collocated entry point

The service "HALocalConnection" provides a collocated entry point to the replica which is the master. On the other replicas, this service blocks the collocated connections until the local replica becomes the master.

```
<service class="org.objectweb.joram.client.jms.ha.local.
                HALocalConnection"/>
```

### 1.3.4 Configuration example

The following configuration defines a HA Joram server "s0" that provides two entry points: Tcp and collocated. It is replicated three times on the same host (localhost).

```
<config>
    <domain name="D1"/>
    <cluster id="0" name="s0">
      <server id="0" hostname="localhost">
        <network domain="D1" port="16301"/>
        <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
                 args="root root"/>
        <service class="org.objectweb.joram.mom.proxies.
                        tcp.TcpProxyService" args="2560"/>
        <service class="org.objectweb.joram.client.jms.ha.local.
                        HALocalConnection"/>
      </server>
      <server id="1" hostname="localhost">
        <network domain="D1" port="16302"/>
        <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
                 args="root root"/>
        <service class="org.objectweb.joram.mom.proxies.
                        tcp.TcpProxyService" args="2561"/>
        <service class="org.objectweb.joram.client.jms.ha.local.
                        HALocalConnection"/>
      </server>
      <server id="2" hostname="localhost">
        <network domain="D1" port="16303"/>
        <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
                 args="root root"/>
        <service class="org.objectweb.joram.mom.proxies.
                        tcp.TcpProxyService" args="2562"/>
        <service class="org.objectweb.joram.client.jms.ha.local.
                        HALocalConnection"/>
      </server>
    </cluster>
</config>
```

# 2 Platform startup

A server replica is started exactly like a standard agent server. The arguments are:

- server id: identifier of the clustered server (not the identifier of the replica)
- storage directory path: path of the directory where the persistency files are created
- replica id: identifier of the replica inside the clustered server

```
java fr.dyade.aaa.agent.AgentServer <server id> <storage directory path>
<replica id>
```

The replicas can be started in any order. The first master replica elected is the first replica connected to the group communication channel.

#### Host clock synchronization

The clocks of the hosts from the cluster should be quite synchronized if the “heart beat” feature is used.

# 3 Programming an external HA Joram client

The only available protocol for an external client is TCP. *The protocol Soap is not yet available.*

## 3.1 Joram administration

### 3.1.1 Create a connection factory

There are several types of HA TCP connection factory depending on the destination reached (generic, queue or topic) and the transactional mode (XA or none).

These connection factories belong to the package *org.objectweb.joram.client.jms.ha.tcp*. To instantiate a connection factory, call the static method *create* defined in the class of the connection factory to instantiate. A parameter must be given specifying the URL of the HA Joram server. This URL defines the list of the TCP entry points owned by each replicas of the HA Joram server.

```
<XA|none><Queue|Topic|none>ConnectionFactory cf =
    org.objectweb.joram.client.jms.ha.tcp.
    <XA|none><Queue|Topic|none>HATcpConnectionFactory.create(
        <HA Joram server URL>);
```

For example to create a generic (not Queue either Topic) non-transactional (no XA) connection factory you need to call:

```
ConnectionFactory cf =
    org.objectweb.joram.client.jms.ha.tcp.HATcpConnectionFactory.create(
    "hajoram://localhost:2560,localhost:2561");
```

#### 3.1.2 Configure the connection factory

It is also necessary to set the *connectionTimer* of the connection according to the switching time required by the Joram server cluster, i.e. the delay needed to elect a new master replica and to activate it. If the *connectionTimer* is too short, then the external client may not be able to connect to the new master replica. So the failure of a Joram server replica may break the HA connection between the external client and the HA Joram server.

```
((org.objectweb.joram.client.jms.ConnectionFactory)cf).getParameters().
    connectingTimer = 30;
```

## 3.2 JMS programming

The JMS programming is completely standard. A HA server behaves exactly the same as a standard server. However there is a slight difference about the naming of the destination. Just like with a standard Joram, you can choose to use JNDI or the internal Joram naming. But if you use JNDI then you have to configure a replicated JNDI server. This operation is available but not documented here as it is simpler to use the Joram internal naming.

# 4 Programming a collocated Joram client

## 4.1 Joram administration

#### 4.1.1 Create a connection factory

There are several types of HA local connection factory depending on the destination reached (generic, queue or topic) and the transactional mode (XA or none).

These connection factories belong to the package *org.objectweb.joram.client.jms.ha.local*. To instantiate a connection factory, call the static method *create* defined in the class of the connection factory to instantiate.

```
<XA|none><Queue|Topic|none>ConnectionFactory cf =
    org.objectweb.joram.client.jms.ha.local.
    <XA|none><Queue|Topic|none>HALocalConnectionFactory.create();
```

For example to create a generic (not Queue either Topic) non-transactional (no XA) connection factory you need to call:

```
ConnectionFactory cf =
    org.objectweb.joram.client.jms.ha.local.
    HALocalConnectionFactory.create();
```

## 4.2 Collocated client process

A collocated client must be instantiated inside the same JVM processes as the Joram server replicas. It also must use the same class loader (or a child) as the server's class loader. A simple way to do this is to write a main class that starts an agent server replica and then creates one or more HA local connections.

In the following code example, a Topic non-transactional (not XA) connection is opened. It is important to notice that the connection creation blocks until the server replica becomes master.

```
public static void main(String[] args) throws Exception {
    AgentServer.init(args);
    AgentServer.start();

    TopicConnectionFactory cf =
      org.objectweb.joram.client.jms.ha.local.
      TopicHALocalConnectionFactory().create();

    // The next line blocks until the server replica becomes master
    TopicConnection cnx = cf.createTopicConnection("root", "root");
```

## 4.3 JMS programming

The JMS programming is completely standard. A HA server behaves exactly the same as a standard server. However there is a slight difference about the naming of the destination. Just like with a standard Joram, you can choose to use JNDI or the internal Joram naming. But if you use JNDI then you have to configure a replicated JNDI server. This operation is available but not documented here as it is simpler to use the Joram internal naming.