Runtime Governance: User Guide

by Gary Brown (Red Hat)

Chapter 1. Overview

This section provides an overview of the Runtime Governance architecture.

The architecture is separated into four distinct areas, with components that bridge between these areas:

- Activity Collector - this component is optional, and can be embedded within an executing environment to manage the collection of information
- Activity Server - this component provides a store and query API for activity information. If not using the Activity Collector, then activity information can be reported directly to the server via a suitable binding (e.g. REST).
- Event Processor Network - this component can be used to analyse the activity information. Each network can be configured with a set of event processing nodes, to filter, transform and analyse the events, to produce relevant rules.
- Active Collection - this component is responsible for maintaining an active representation of information being collected. UI components can then access this information via REST services to present the information to users (e.g. via gadgets)

This document will explain how a user can configure these components to work together to build a Runtime Governance solution to realtime monitoring of executing business transactions.

Chapter 2. Installation

This section will describe how to install Overlord Runtime Governance in different environments.

2.1. JBoss Application Server

This section describes how to install Overlord Runtime Governance into the JBoss Application Server.

2.1.1. Install

Download the latest release from the Overlord Runtime Governance website, selecting the distribution specific to JBoss AS. Then unpack the distribution into a suitable location.

Make sure that the JBOSS_HOME environment variable is set to the root folder of the JBoss AS installation.

The final step is to perform the installation using maven. You will need maven 3.0.4 or higher, and can be downloaded from here: *http://maven.apache.org/download.html*

To do the installation, use the following command from the root folder of the installation:

```
mvn install [ -Dtype=<installation-type> ]
```

The *installation-type* value can be:

Value	Description
server	This will result in the full server configuration being installed into the server. This is the default value.
restc	This will result in only the activity collector functionality being installed, using a RESTful client to communicate with a remote Runtime Governance server.

2.1.2. Configuration

The configuration for the Runtime Governance capability within a JBoss AS environment can be found in the file `$JBOSS_HOME/standalone/configuration/overlord-rtgov.properties`. The particular properties of interest will depend upon the installation type.

2.1.2.1. "server" Installation Type

When installing the full Runtime Governance server, modification to the configuration will generally only be necessary if running in a clustered environment and/or wishing to use a particular database. The default installed environment is intended to work "out of the box", and uses an in-memory H2 database.

However, specific technologies used in the Activity Server, Event Processor Network or Active Collection modules may need to use different configuration properties to work correctly within a clustered environment. More details will be provided in sections discussing those technologies, however here we will present the common changes that may be required.

Database

The database is defined by the configuration in two places:

(i) The datasource is configured in the `$JBOSS_HOME/standalone/deployment/overlord-rtgov/rtgov-ds.xml`.

(ii) Properties supplied to the Entity Manager are configured in the `overlord-rtgov.properties` file.

Caching

The EPN and Active Collection mechanisms both have the ability to make use of caching provided by infinispan. When running the AS7 server in clustered mode (i.e. with `standalone-full-ha.xml`), the server provides a default clustered cache container, which is referenced in the *infinispan.container* property in the `overlord-rtgov.properties` file. Simply uncomment this property to enable the EPN and Active Collection Source configurations that do not explicitly provide a container JNDI name, to make use of this default clustered cache container.

However, to make sure the individual named caches are clustered correctly, it is necessary to add an entry for each cache into the `standalone-full-ha.xml` file. As an example, the following cache entry for the "Principals" cache has been defined, for use with the Policy Enforcement example:

```
            <cache-container name="cluster" aliases="ha-partition" default-
cache="default">
                <transport lock-timeout="60000"/>
                <replicated-cache name="default" mode="SYNC"
 batching="true">
                    <locking isolation="REPEATABLE_READ"/>
                </replicated-cache>

                <!-- Configuration for Runtime Governance caches -->

                <replicated-cache name="Principals" mode="SYNC">
                    <locking isolation="REPEATABLE_READ"/>
                    <transaction mode="FULL_XA" locking="PESSIMISTIC"/>
                </replicated-cache>
            </cache-container>
```

2.1.2.2. "restc" Installation Type

This installation type is used to configure an execution environment that will be sending its activity information to a remote Runtime Governance server using REST.

Before being able to use this Runtime Governance client, it will be necessary to configure the serverURL property in the `overlord-rtgov.properties` file, to point to the Runtime Governance server.

2.1.3. Uninstall

To uninstall, simply perform the following command in the root folder of the installation, ensuring that the JBOSS_HOME environment variable refers to the root location of the JBoss AS environment:

```
mvn clean
```

Chapter 3. Reporting Activity Information

There are two ways in which activity information can be collected for further processing by the Runtime Governance server.

1. Integrating an *activity collector* into the execution environment. This will intercept activities and automatically report them to the Runtime Governance server.
2. Manually report the activity information to the Runtime Governance server through a publicly available API (e.g. REST service)

This section will explain how to use both approaches.

3.1. Embedded Activity Collector

3.1.1. Execution Environments

This section describes how activities can be collected from different execution environments.

3.1.1.1. SwitchYard

To instrument a switchyard application, that is deployed as a war, is simply a case of including a maven dependency and configuring a manifest property within the built war file.

The maven dependency added to the pom.xml for the SwitchYard project is:

```
                <dependency>
                        <groupId>org.overlord.rtgov.integration</groupId>
                        <artifactId>rtgov-switchyard</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
```

and the following build plugin, to include the dependency between the SwitchYard application and the Overlord Runtime Governance infrastructure:

```
    <build>
        <plugins>
            ...
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <!-- Java EE 6 doesn't require web.xml, Maven needs to
catch up! -->
                    <failOnMissingWebXml>false</failOnMissingWebXml>
```

```
                    <webResources>
                        <resource>
                            <directory>target/switchyard_xml</directory>
                        </resource>
                    </webResources>
                    <archive>
                        <manifestEntries>
                            <Dependencies>deployment.overlord-rtgov.war</
Dependencies>
                        </manifestEntries>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

3.1.2. Activity Interceptor

The Activity Interceptor mechanism provides the means to install event processing capabilities within the activity collection environment (i.e. co-located with the execution of the business transaction).

The main reason for performing analysis of the activity events at this stage in the runtime governance lifecycle is to enable the analysis to potential block the business transaction. For an example of such a case, please see the synchronous policy sample.

3.1.2.1. Defining the Activity Interceptors

The Activity Interceptor can be defined as an object model or specified as a JSON representation for packaging in a suitable form, and subsequently de-serialized when deployed to the governed execution environment.

The following is an example of the JSON representation of a list of Activity Interceptors. This particular example is from the synchronous policy sample:

```
[{
  "name" : "RestrictUsage",
  "version" : "1",
  "predicate" : {
    "@class" : "org.overlord.rtgov.ep.mvel.MVELPredicate",
    "expression" : "event instanceof
 org.overlord.rtgov.activity.model.soa.RequestReceived && event.serviceType
 == \"{urn:switchyard-quickstart-demo:orders:0.1.0}OrderService\""
  },
  "eventProcessor" : {
    "@class" : "org.overlord.rtgov.ep.mvel.MVELEventProcessor",
    "script" : "VerifyLastUsage.mvel",
    "services" : {
      "CacheManager" : {
```

```
        "@class" :
 "org.overlord.rtgov.ep.service.infinispan.InfinispanCacheManager"
      }
    }
  }
}]
```

This example illustrates the configuration of a single Activity Interceptor with the top level elements:

Field	Description
name	The name of the Activity Interceptor.
version	The version of the Activity Interceptor. If multiple versions of the same named Activity Interceptor are installed, only the newest version will be used. Versions can be expressed using three schemes: Numeric - i.e. simply define the version as a number Dot Format - i.e. 1.5.1.Final Any alpha, numeric and symbols.
predicate	The optional implementation of the `org.overlord.rtgov.ep.Predicate` interface, used to determine if the activity event is relevant and therefore should be supplied to the event processor
eventProcessor	The implementation of the `org.overlord.rtgov.ep.EventProcessor` interface, that is used to analyse the activity event

When comparing versions, for example when determining whether a newly deployed Activity Interceptor has a higher version than an existing one with the same name, then initially the versions will be compared as numeric values. If either are not numeric, then they will be compared using dot format, with each field being compared first as numeric values, and if not based on lexical comparison. If both fields don't have a dot, then they will just be compared lexically.

3.1.2.2. Registering the Activity Interceptors

JEE Container

The Activity Interceptors are deployed within the JEE container as a WAR file with the following structure:

```
warfile
```

```
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-ai.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-ai-loader-jee.jar
|       |-<additional libraries>
```

The `ai.json` file contains the JSON representation of the Activity Interceptor configuration.

The `ai-loader-jee.jar` acts as a bootstrapper to load and register the Activity Interceptors.

If custom classes are defined, then the associated classes and resources can be defined in the `WEB-INF/classes` folder or within additional libraries located in the `WEB-INF/lib` folder.

A maven pom.xml that will create this structure is:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-management</
groupId>
                        <artifactId>activity</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-management</
groupId>
                        <artifactId>ai-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
```

```
                </dependency>
                ....
        </dependencies>

</project>
```

If deploying in JBoss Application Server, then the following fragment also needs to be included, to define the dependency on the core Overlord Runtime Governance modules:

```
.....
        <build>
                <finalName>....</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-plugin</artifactId>
                                <configuration>
                                        <failOnMissingWebXml>false</
failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

 <Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

3.1.3. Information Processor

To enable the Runtime Governance infrastructure, and the user policies/rules that are defined within it, to make the most effective use of the activities that are reported, it is necessary to process certain events to extract relevant information for use in:

- correlating activity events to a particular business transaction instance
- highlighting important properties that may need to be used in business policies

It is also important to control what information is distributed with the actvity events, for both size (i.e. performance) and security reasons. By default information content should not be distributed, unless an information processor has been defined to explicitly indicate how that information should be represented (if at all) within the activity event.

This section explains how information processors can be configured and deployed along side the business applications they are monitoring.

3.1.3.1. Defining the Information Processors

The Information Processor can be defined as an object model or specified as a JSON representation for packaging in a suitable form, and subsequently de-serialized when deployed to the governed execution environment.

The following is an example of the JSON representation of a list of Information Processors. This particular example accompanies the Order Management sample:

```
[{
        "name":"OrderManagementIP",
        "version":"1",
        "typeProcessors":{
                "{urn:switchyard-quickstart-demo:orders:1.0}submitOrder":{
                        "contexts":[{
                                "type":"Conversation",
                                "evaluator":{
                                        "type":"xpath",
                                        "namespaces":{
                                                "orders":"urn:switchyard-
quickstart-demo:orders:1.0"
                                        },
                                        "expression":"/orders:submitOrder/
order/orderId"
                                }
                        }],
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"xpath",
                                        "namespaces":{
                                                "orders":"urn:switchyard-
quickstart-demo:orders:1.0"
                                        },
                                        "expression":"/orders:submitOrder/
order/customer"
                                }
                        }]
                },
                "java:org.switchyard.quickstarts.demos.orders.OrderAck":{
                        "contexts":[{
                                "type":"Conversation",
                                "evaluator":{
                                        "type":"mvel",
                                        "expression":"orderId"
                                }
                        }],
                        "properties":[{
                                "name":"customer",
```

```
                                "evaluator":{
                                        "type":"mvel",
                                        "expression":"customer"
                                }
                        },{
                                "name":"total",
                                "evaluator":{
                                        "type":"mvel",
                                        "expression":"total"
                                }
                        }]
                },
                "{urn:switchyard-quickstart-demo:orders:1.0}makePayment":{
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"xpath",
                                        "namespaces":{
                                                "orders":"urn:switchyard-
quickstart-demo:orders:1.0"
                                        },
                                        "expression":"/orders:makePayment/
payment/customer"
                                }
                        },{
                                "name":"amount",
                                "evaluator":{
                                        "type":"xpath",
                                        "namespaces":{
                                                "orders":"urn:switchyard-
quickstart-demo:orders:1.0"
                                        },
                                        "expression":"/orders:makePayment/
payment/amount"
                                }
                        }]
                },
                "java:org.switchyard.quickstarts.demos.orders.Receipt":{
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"mvel",
                                        "expression":"customer"
                                }
                        },{
                                "name":"amount",
                                "evaluator":{
                                        "type":"mvel",
                                        "expression":"amount"
```

```
                                }
                        }]
                },

  "java:org.switchyard.quickstarts.demos.orders.ItemNotFoundException":{
                        "script":{
                                "type":"mvel",
                                "expression":"activity.fault =
  \"ItemNotFound\""
                        }
                }
        }
}]
```

This example illustrates the configuration of a single Information Processor with the top level elements:

Field	Description
name	The name of the Information Processor.
version	The version of the Information Processor. If multiple versions of the same named Information Processor are installed, only the newest version will be used. Versions can be expressed using three schemes: Numeric - i.e. simply define the version as a number Dot Format - i.e. 1.5.1.Final Any alpha, numeric and symbols.
typeProcesors	The map of type processors - one per type, with the type name being the map key.

When comparing versions, for example when determining whether a newly deployed Information Processor has a higher version than an existing one with the same name, then initially the versions will be compared as numeric values. If either are not numeric, then they will be compared using dot format, with each field being compared first as numeric values, and if not based on lexical comparison. If both fields don't have a dot, then they will just be compared lexically.

Type Processor

The type processor element is associated with a particular information type (i.e. as its key). The fields associated with this component are:

Field	Description
contexts	The list of context evaluators.

Field	Description
properties	The list of property evaluators.
script	An optional script that is used to do any other processing that may be required.
transformer	An optional transformer that determines how this information type will be represented within an activity event.

Context Evaluator

The fields associated with the Context Evaluator component are:

Field	Description
type	The context type, e.g. Conversation, Endpoint or Message.
expression	The expression evaluator used to derived the context value. See further down for details.
optional	Optional field that indicates whether the value being extracted by the expression is optional. The default is false. If a value is not optional, but the expression fails to locate a value, then an error will be reported

Property Evaluator

The fields associated with the Property Evaluator component are:

Field	Description
name	The property name being initialized.
expression	The expression evaluator used to derive the property value. See further down for details.
optional	Optional field that indicates whether the value being extracted by the expression is optional. The default is false. If a value is not optional, but the expression fails to locate a value, then an error will be reported

Expression Evaluator

In the context and property evaluator components, they reference an expression evaluator that is used to derive their value. The expression evaluator has the following fields:

Field	Description
type	The type of expression evaluator to use. Currently only support **mvel** or **xpath**.

Field	Description
expression	The expression to evaluate.

These expressions operate on the information being processed, to return a string value to be applied to the appropriate context or property.

Script

The script field of the Type Processor has the following fields:

Field	Description
type	The type of script evaluator to use. Currently only support **mvel**.
expression	The expression to evaluate.

The MVEL script evaluator is supplied two variables for its use:

- information - The information being processed
- activity - The activity event

An example of how this script can be used is shown in the example above, associated with the *ItemNotFoundException*. In this case, the message on the wire does not carry the fault name, so the information processor is used to set the *fault* field on the activity event.

Transformer

The transformer field of the Type Processor has the following fields:

Field	Description
type	The type of transformer to use. Currently support **serialize** and **mvel**.

The *serialize* transformer does not take any other properties. It simply attempts to convert the representation of the information into a textual form for inclusion in the activity event. So this transformer type can be used where the complete information content is required.

The *mvel* transformer takes the following additional fields:

The MVEL transformer script is supplied the following variable for its use:

Field	Description
expression	The mvel expression to transform the supplied information.

The MVEL transformer is supplied the following variable for its use:

- information - The information being processed

For example, to include the content of the submitOrder message:

```
        "typeProcessors":{
                "{urn:switchyard-quickstart-demo:orders:1.0}submitOrder":{
                        "contexts":[{
                                "type":"Conversation",
                                "evaluator":{
                                        "type":"xpath",
                                        "namespaces":{
                                                "orders":"urn:switchyard-
quickstart-demo:orders:1.0"
                                        },
                                        "expression":"/orders:submitOrder/
order/orderId"
                                }
                        }],
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"xpath",
                                        "namespaces":{
                                                "orders":"urn:switchyard-
quickstart-demo:orders:1.0"
                                        },
                                        "expression":"/orders:submitOrder/
order/customer"
                                }
                        }],
                        "transformer":{
                                "type":"serialize"
                        }
                },
```

3.1.3.2. Registering the Information Processors

JEE Container

The Information Processors are deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
```

```
|    |    |-ip.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-ip-loader-jee.jar
|       |-<additional libraries>
```

The `ip.json` file contains the JSON representation of the Information Processor configuration.

The `ip-loader-jee.jar` acts as a bootstrapper to load and register the Information Processors.

If custom classes are defined, then the associated classes and resources can be defined in the `WEB-INF/classes` folder or within additional libraries located in the `WEB-INF/lib` folder.

A maven pom.xml that will create this structure is:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-management</
groupId>
                        <artifactId>activity</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-management</
groupId>
                        <artifactId>ip-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                ....
        </dependencies>

</project>
```

If deploying in JBoss Application Server, then the following fragment also needs to be included, to define the dependency on the core Overlord Runtime Governance modules:

```
.....
        <build>
                <finalName>....</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-plugin</artifactId>
                                <configuration>
                                        <failOnMissingWebXml>false</
failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

 <Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

3.2. Reporting and Querying Activity Events via REST

This section explains how activity information can be reported to, and queried from, the Activity Server via a RESTful service.

3.2.1. Reporting Activity Information

POST request to URL: `<host>/overlord-rtgov/activity/store`

The request contains the list of ActivityUnit objects encoded in JSON. For example,

```
[{
    "id":"TestId1",
    "activityTypes":[{
        "type":"RequestSent",
        "context":[{
            "value":"12345"
        },{
            "value":"abc123",
            "type":"Endpoint"
        },{
            "value":"ABC123",
            "type":"Message"
        }],
```

```
        "content":"....",
        "serviceType":"{http://service}OrderService",
        "operation":"buy",
        "fault":"MyFault",
        "messageType":"{http://message}OrderRequest",
        "timestamp":1347028592880
    },{
        "type":"ResponseReceived",
        "context":[{
            "value":"12345"
        },{
            "value":"ABC124",
            "type":"Message"
        }],
        "content":"....",
        "serviceType":"{http://service}OrderService",
        "operation":"buy",
        "fault":"OutOfStock",
        "messageType":"{http://message}OutOfStock",
        "replyToId":"ABC123",
        "timestamp":1347028593010
    }],
    "origin":{
        "host":"Saturn",
        "port":"8010",
        "principal":"Fred",
        "node":"Saturn1",
        "thread":"Thread-1"
    }
},{
    .....
}]
```

3.2.2. Querying Activity Events using an Expression

POST request to URL: `<host>/overlord-rtgov/activity/query`

The request contains the JSON encoding of the Query Specification, which has the following properties:

Property	Description
id	Optionally specifies the activity unit id that is required.
fromTimestamp	Optionally specifies the start date/time for the activity units required. If not specified, then the query will apply to activity units from the first one recorded.

Property	Description
toTimestamp	Optionally specifies the end date/time for the activity units required. If not specified, then the query will relate up to the most recently recorded activity units.
expression	An optional expression that can be used to specify the activity events of interest.
format	Optionally specifies the format of the expression. The value must be supported by the configured activity store.

The response contains a list of ActivityType objects encoded in JSON, which would be similar in form to the example shown above when recording a list of activity units.

3.2.3. Retrieving an Activity Unit

GET request to URL: `<host>/overlord-rtgov/activity/unit?id=<unitId>`

The <unitId> represents the identifier associated with the ActivityUnit that is being retrieved.

3.2.4. Retrieve Activity Events associated with a Context Value

GET request to URL: `<host>/overlord-rtgov/activity/events?context=<identifier>`

The <identifier> represents the correlation value associated with the ActivityType(s) that are being retrieved.

Chapter 4. Analyzing Events

4.1. Configuring an Event Processor Network

An Event Processor Network is a mechanism for processing a stream of events through a network of linked nodes established to perform specific filtering, transformation and/or analysis tasks.

4.1.1. Defining the Network

The network can be defined as an object model or specified as a JSON representation for packaging in a suitable form, and subsequently de-serialized when deployed to the runtime governance server.

The following is an example of the JSON representation of an Event Processor Network. This particular example defines the "out of the box" EPN installed with the distribution:

```
{
  "name" : "Overlord-RTGov-EPN",
  "version" : "1.0.0-SNAPSHOT",
  "subscriptions" : [ {
    "nodeName" : "SOAEvents",
    "subject" : "ActivityUnits"
  },
  {
    "nodeName" : "ServiceDefinitions",
    "subject" : "ActivityUnits"
  } ],
  "nodes" : [
    {
      "name" : "SOAEvents",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ "SOAEvents" ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" :
 "org.overlord.rtgov.content.epn.SOAActivityTypeEventSplitter"
      },
      "predicate" : null,
      "notifications" : [ ]
    },{
      "name" : "ServiceDefinitions",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
```

```
        "@class" :
 "org.overlord.rtgov.content.epn.ServiceDefinitionProcessor"
      },
      "predicate" : null,
      "notifications" : [ {
        "type" : "Results",
        "subject" : "ServiceDefinitions"
      } ]
    },{
      "name" : "ServiceResponseTimes",
      "sourceNodes" : [ "ServiceDefinitions" ],
      "destinationSubjects" : [ "ServiceResponseTimes" ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" :
 "org.overlord.rtgov.content.epn.ServiceResponseTimeProcessor"
      },
      "predicate" : null,
      "notifications" : [ {
        "type" : "Results",
        "subject" : "ServiceResponseTimes"
      } ]
    }
  ]
}
```

Another example of a network, used within one of the quickstarts is:

```
{
  "name" : "AssessCreditPolicyEPN",
  "version" : "1",
  "subscriptions" : [ {
    "nodeName" : "AssessCredit",
    "subject" : "SOAEvents"
  } ],
  "nodes" : [
    {
      "name" : "AssessCredit",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "predicate" : {
        "@class" : "org.overlord.rtgov.ep.mvel.MVELPredicate",
        "expression" : "event.serviceProvider && !event.request
 && event.serviceType == \"{urn:switchyard-quickstart-
demo:orders:0.1.0}OrderService\""
      },
```

```
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.mvel.MVELEventProcessor",
        "script" : "AssessCredit.mvel",
        "services" : {
          "CacheManager" : {
            "@class" :
 "org.overlord.rtgov.ep.service.infinispan.InfinispanCacheManager"
          }
        }
      }
    }
  ]
}
```

This example illustrates the configuration of a service associate with the event processor, as well as a predicate.

The top level elements of this descriptior are:

Field	Description
name	The name of the network.
subscriptions	The list of subscriptions associated with the network, discussed below.
nodes	The nodes that form the connected graph within the network, discussed below.
version	The version of the network. Versions can be expressed using three schemes: Numeric - i.e. simply define the version as a number Dot Format - i.e. 1.5.1.Final Any alpha, numeric and symbols

When comparing versions, for example when determining whether a newly deployed EPN has a higher version than an existing network with the same name, then initially the versions will be compared as numeric values. If either are not numeric, then they will be compared using dot format, with each field being compared first as numeric values, and if not based on lexical comparison. If both fields don't have a dot, then they will just be compared lexically.

4.1.1.1. Subscription

The subscription element is used to define a subject that the network is interested in, and the name of the node to which the events from that subject should be routed.

This decoupled subscription approach enables multiple networks to register their interest in events from the same subject. Equally multiple nodes within the same network could subscribe to the same subject.

The fields associated with this component are:

Field	Description
Subject	The subject to subscribe to.
nodeName	The name of the node within the network to route the events to.

Reserved subjects

This is a list of the subjects that are reserved for Overlord's use:

Subject	Purpose
ActivityUnits	This subject is used to publish events of the type org.overlord.rtgov.activity.model.ActivityUnit, produced when activity information is recorded with the Activity Server.

4.1.1.2. Node

This element is used to define a particular node in the graph that forms the network, and has the following fields:

Field	Description
name	The name of the node.
sourceNodes	A list of node names that represent the source nodes, within the same network, that this node receives its events from. Therefore, if this list is empty, it means that the node is a *root* node and should be the target of a subscription.
destinationSubjects	A list of inter-EPN subjects to publish any resulting events to. Note: these subjects are only of relevance to other networks.
maxRetries	The maximum number of times an event should be retried, following a failure, before giving up on the event.
retryInterval	The delay that should occur between retry attempts - may only be supported in some environments.
eventProcessor	Defines the details for the event processor implementation being used. At a minimum, the value for this field should define a @class property to specify the Java class name for

Field	Description
	the event process implementation to use. Another general field that can be configured is the map of services that can be used by the event processor. Depending upon which implementation is selected, the other fields within the value will apply to the event processor implementation.
predicate	This field is optional, but if specified will define a predicate implementation. As with the event processor, it must at a minimum define a @class field that specifies the Java class name for the implementation, with any additional fields be used to initialize the predicate implementation.
notifications	A list of notifications. A notification entry will define its **type** (explained below) and the notification **subject** upon which the information should be published. Unlike the *destinationSubjects* described above, which are subjects for inter-EPN communication, these notification subjects are the mechanism for distribution information out of the EPN capability, for presentation to end-users through various means.

Notify Types

The *notify types* field defines what type of notifications should be emitted from a node when processing an event. The notifications are the mechanism used by potentially interested applications to observe what information each node is processing, and the results they produce.

The possible values for this field are:

Field	Description
Processed	This type indicates that a notification should be created when an event is considered suitable for processing by the node. An event is suitable either if no predicate is defined, or if the predicate indicates the event is valid.
Results	This type indicates that a notification should be created for any information produced as the result of the event processor processing the event.

Tip

Notifications are the mechanism for making information processed by the Event Processor Network accessible by interested parties. If a notity type(s) is not defined for a node, then it will only be used for internal processing, potentially supplying the processed event to other nodes in the network (or other networks if destination subject(s) are specified).

4.1.2. Registering the Network

4.1.2.1. JEE Container

The Event Processor Network is deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-epn.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-epn-loader-jee.jar
|       |-<additional libraries>
```

The `epn.json` file contains the JSON representation of the EPN configuration.

The `epn-loader-jee.jar` acts as a bootstrapper to load and register the Event Processor Network.

If custom predicates and/or event processors are defined, then the associated classes and resources can be defined in the `WEB-INF/classes` folder or within additional libraries located in the `WEB-INF/lib` folder.

A maven pom.xml that will create this structure is:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
```

```
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.event-processor-
network</groupId>
                        <artifactId>epn-core</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.event-processor-
network</groupId>
                        <artifactId>epn-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                ....
        </dependencies>

</project>
```

If deploying in JBoss Application Server, then the following fragment also needs to be included, to define the dependency on the core Overlord Runtime Governance modules:

```
.....
        <build>
                <finalName>slamonitor-epn</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-plugin</artifactId>
                                <configuration>
                                        <failOnMissingWebXml>false</
failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

 <Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
```

```
        </build>
        .....
```

4.1.3. Supporting Multiple Versions

Event Processor Networks define a version number that can be used to keep track of the evolution of changes in a network.

When a network is deployed to a container, and used to process events, a newer version of the network can be deployed along side the existing version to ensure there is continuity in the processing of the event stream. New events presented to the network will be processed by the most recent version, while events still being processed by a particular version of the network, will continue to be processed by the same version - thus ensuring that changes to the internal structure of the network do not impact events that are mid-way through being processed by the network.

The management features, discussed later in the User Guide, can be used to determine when an older version of the network last processed an event - and therefore when an older version has been inactive for a suitable amount of time, it can be unregistered.

4.2. Event Processors

Although custom event processors can be defined, there are some "out of the box" implementations. These are discussed in the following sub-sections.

4.2.1. Drools Event Processor

The Drools Event Processor implementation (`org.overlord.rtgov.epn.drools.DroolsEventProcessor`) enables events to be processed by a Complex Event Processing (CEP) rule. This implementation defines the following additional fields:

Field	Description
ruleName	The name of the rule, used to locate the rule definition in a file called "<ruleName>.drl".

An example of such a rule is:

```
import org.overlord.rtgov.activity.model.soa.RequestReceived
import org.overlord.rtgov.activity.model.soa.ResponseSent

global org.overlord.rtgov.ep.EPContext epc

declare RequestReceived
    @role( event )
    @timestamp( timestamp )
    @expires( 2m20s )
```

```
end

declare ResponseSent
    @role( event )
    @timestamp( timestamp )
    @expires( 2m20s )
end

rule "correlate request and response"
when
    $req : RequestReceived( $id : messageId ) from entry-point "Purchasing"
    $resp : ResponseSent( replyToId == $id, this after[0,2m20s] $req )  from
 entry-point "Purchasing"
then

    epc.logInfo("REQUEST: "+$req+" RESPONSE: "+$resp);

    java.util.Properties props=new java.util.Properties();
    props.put("requestId", $req.getMessageId());
    props.put("responseId", $resp.getMessageId());

    long responseTime=$resp.getTimestamp()-$req.getTimestamp();

    epc.logDebug("CORRELATION on id '"+$id+"' response time "+responseTime);

    props.put("responseTime", responseTime);

    epc.handle(props);

end
```

This is an example of a rule used to correlate request and response events. When a correlation is found, then a ResponseTime object is created and "forwarded" to the Event Processor Network for further processing using the *handle* method.

The source of the events into the rule are named entry points, where the name relates to the source node or subject that supplies the events.

The rule has access to external capabilities through the *EPContext*, which is defined in the statements:

```
global org.overlord.rtgov.ep.EPContext epc
```

which is used at the end of the above example to handle the result of the event processing (i.e. to forward a derived event back into the network).

If an error occurs, that requires the event to be retried (within the Event Processor Network), or the business transaction blocked (when used as a synchronous policy), then the rule can either throw an exception or return the exception as the result using the *handle()* method.

Caution

Temporal rules do not currently work in a clustered environment. This is because correlation between events occurs in working memory, which is not shared across servers. Therefore for the correlation to work, all relevant events must be received by a single server.

4.2.2. MVEL Event Processor

A MVEL based Event Processor implementation (`org.overlord.rtgov.epn.mvel.MVELEventProcessor`) enables events to be processed by a MVEL script. This implementation defines the following additional fields:

Field	Description
script	The location of the MVEL script, which may be relative to the classpath.

The script will have access to the following variables:

Variable	Description
source	The name of the source node or subject upon which the event was received.
event	The event to be processed.
retriesLeft	The number of retries remaining.
epc	The EP context (`org.overlord.rtgov.ep.EPContext`), providing some utility functions for use by the script, including the *handle* method for pushing the result back into the network.

If an error occurs, that requires the event to be retried (within the Event Processor Network), or the business transaction blocked (when used as a synchronous policy), then the script can return the exception as the result using the *handle()* method.

4.2.3. Supporting Services

This section describes a set of supporting services available to some of the Event Processor implementations. See the documentation for the specific Event Processor implementations for information on how to access these services.

4.2.3.1. Cache Manager

Description

The purpose of the Cache Manager service is to enable event processors to store and retrieve information in named caches.

API

Method	Description
<K,V> Map<K,V> getCache(String name)	This method returns the cache associated with the supplied name. If the cache does not exist, then a null will be returned.
boolean lock(String cacheName, Object key)	This method locks the item, associated with the supplied key, in the named cache.

Implementations

Embedded

Class name: `org.overlord.rtgov.ep.service.InMemoryCacheManager`

This class provides a transient in-memory implementation of the cache manager. This implementation does not support locking, so will return *true* to all lock requests.

Infinispan

Class name: `org.overlord.rtgov.ep.service.infinispan.InfinispanCacheManager`

This class provides an implementation based on Infinispan. The properties for this class are:

Property	Description
container	The optional JNDI name for the infinspan container defined in the `standalone-full.xml` or `standalone-full-ha.xml` file.

The container will be obtained in three possible ways.

(a) if the container is explicitly defined, then it will be used

(b) if the container is not defined, then a default container will be obtained from the `$JBOSS_HOME/standalone/configuration/overlord-rtgov.properties` file for the infinispan.container property.

(c) if no default container is defined, then an embedded cache manager will be created based on the configuration obtained from the `rtgov_infinispan.xml` file contained in the `overlord-rtgov.war`.

4.3. Predicates

Although custom event processors can be defined, there are some "out of the box" implementations:

4.3.1. MVEL Predicate

A MVEL based Predicate implementation (`org.overlord.rtgov.epn.mvel.MVELPredicate`) enables events to be evaluated by a MVEL expression or script. This implementation defines the following additional fields:

Field	Description
expression	The MVEL expression used to evaluate the event.
script	The location of the MVEL script, which may be relative to the classpath.

Caution

Only the expression or script should be defined, not both.

The expression or script will have access to the following variables:

Variable	Description
event	The event to be processed.

Chapter 5. Accessing Derived Information

5.1. Configuring Active Colletions

An Active Collection is similar to a standard collection, but with the ability to report change notifications when items are inserted, updated or removed. The other main difference is that they cannot be directly updated - their contents is managed by an Active Collection Source which acts as an adapter between the collection and the originating source of the information.

This section will explain how to define an Active Collection Source and register it to indirectly create an Active Collection.

5.1.1. Defining the Source

The source can be defined as an object model or specified as a JSON representation for packaging in a suitable form, and subsequently de-serialized when deployed to the runtime governance server.

The following is an example of the JSON representation that defines a list of Active Collection Sources - so more than one source can be specified with a single configuration:

```
[
  {
    "@class" :
 "org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "ServiceResponseTimes",
    "type" : "List",
    "itemExpiration" : 0,
    "maxItems" : 100,
    "subject" : "ServiceResponseTimes",                          //
 Attribute specific to the source implementation
    "aggregationDuration" : 1000,
    "groupBy" : "serviceType + \":\" + operation + \":\" + fault",
    "aggregationScript" : "AggregateServiceResponseTime.mvel"
  },{
    "@class" :
 "org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "ServiceDefinitions",
    "type" : "Map",
    "itemExpiration" : 0,
    "maxItems" : 100,
    "subject" : "ServiceDefinitions",
    "scheduledScript" : "TidyServiceDefinitions.mvel",
    "scheduledInterval" : 60000,
```

```
    "properties" : {
        "maxSnapshots" : 5
    },
    "maintenanceScript" : "MaintainServiceDefinitions.mvel"
  },{
    "@class" :
"org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "Situations",
    "type" : "List",
    "itemExpiration" : 40000,
    "maxItems" : 0,
    "subject" : "Situations",                                    //
Attribute specific to the source implementation
    "activeChangeListeners" : [ {
        "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
        "objectName" : "overlord.rtgov:name=Situations",
        "descriptionScript" : "SituationDescription.mvel",
        "insertTypeScript" : "SituationType.mvel"
    } ]
  },{
    "@class" :
"org.overlord.rtgov.active.collection.ActiveCollectionSource",
    "name" : "Principals",
    "type" : "Map",
    "lazy" : true,
    "visibility" : "Private",
    "factory" : {
        "@class" :
"org.overlord.rtgov.active.collection.infinispan.InfinispanActiveCollectionFactory",
        "cache" : "Principals"
    }
  }
]
```

This configuration shows the definition of two Active Collection Sources. The top level elements for a source, that are common to all active collection sources, are:

Field	Description
@class	This attribute defines the Java class implementing the Active Collection Source. This class must be directly or indirectly derived from `org.overlord.rtgov.active.collection.ActiveCollectionSo`
name	The name of the source and also associated Active Collection.
type	The type of active collection. The currently supported values (as defined in the

Field	Description
	`org.overlord.rtgov.active.collection.ActiveCollectionTy` enum are: List (default) Map
visibility	The visibility of active collection, i.e. whether accessible via the remote access mechanisms such as REST. The currently supported values (as defined in the `org.overlord.rtgov.active.collection.ActiveCollectionVi` enum are: Public (default) Private
lazy	Whether active collection should be created on startup, or lazily instantiated upon first use. The default is false.
itemExpiration	If not zero, then defines the number of milliseconds until an item in the collection should expire (i.e. be removed).
maxItems	If not zero, defines the maximum number of items that the collection should hold. If an insertion causes the size of the collection to increase above this value, then the oldest item should be removed.
aggregationDuration	The duration (in milliseconds) over which the information will be aggregated.
groupBy	An expression defining the key to be used to categorize the information being aggregated. The expression can use properties associated with the information being aggregated.
aggregationScript	The MVEL script to be used to aggregated the information. An example will be shown in a following sub-section.
scheduledInterval	The interval (in milliseconds) between the invocation of the scheduled script.
scheduledScript	The MVEL script invoked at a fixed interval to perform routine tasks on the collection.
maintenanceScript	By default, events received by the active collection source will be inserted into the

Field	Description
	associated active collection. If a MVEL maintenance script is specified, then it will be invoked to manage the way in which the received information will be applied to the active collection.
properties	A set of properties that can be access by the various scripts.
activeChangeListeners	The list of active change listeners that should be instantiated and automatically registered with the Active Collection. The listeners must be derived from the Java class `org.overlord.rtgov.active.collection.AbstractActiveChar`
factory	The optional factory for creating the active collection, derived from the class `org.overlord.rtgov.active.collection.ActiveCollectionFa`

The additional attributes associated with the `EPNActiveCollectionSource` implementation will be discussed in a later section.

5.1.1.1. Scripts

The aggregation script is used to (as the name suggests) aggregate information being provided by the source, before being applied to the collection. The values available to the MVEL script are:

Variable	Description
events	The list of events to be aggregated.

The scheduled script is used to perform regular tasks on the active collection, independent of any information being applied to the collection. The values available to the MVEL script are:

Variable	Description
acs	The active collection source.
acs.properties	The properties configured for the active collection source.
variables	A map associated with the active collection source that can be used by the scripts to cache information.

The maintenance script is used to manage how new information presented to the source is applied to the active collection. If no script is defined, then the information will be inserted by default. The values available to the MVEL script are:

Variable	Description
acs	The active collection source.
acs.properties	The properties configured for the active collection source.
key	The key for the information being inserted. May be null.
value	The value for the information being inserted.
variables	A map associated with the active collection source that can be used by the scripts to cache information.

An example script, showing how these variables can be used is:

```
int maxSnapshots=acs.properties.get("maxSnapshots");

snapshots = variables.get("snapshots");

if (snapshots == null) {
        snapshots = new java.util.ArrayList();
        variables.put("snapshots", snapshots);
}

// Update the current snapshot
currentSnapshot = variables.get("currentSnapshot");

if (currentSnapshot == null) {
        currentSnapshot = new java.util.HashMap();
}

snapshots.add(new java.util.HashMap(currentSnapshot));

currentSnapshot.clear();

// Remove any snapshots above the number configured
while (snapshots.size() > maxSnapshots) {
        snapshot = snapshots.remove(0);
}

// Merge snapshots
merged =
 org.overlord.rtgov.analytics.service.util.ServiceDefinitionUtil.mergeSnapshots(snapshots);

// Update existing, and remove definitions no longer relevant
foreach (entry : acs.activeCollection) {
        org.overlord.rtgov.analytics.service.ServiceDefinition sd=null;

        if (merged.containsKey(entry.key)) {
```

```
                acs.update(entry.key, merged.get(entry.key));
        } else {
                acs.remove(entry.key, entry.value);
        }

        merged.remove(entry.key);
}

// Add new definitions
for (key : merged.keySet()) {
        acs.insert(key, merged.get(key));
}
```

This example shows the script accessing the Active Collection Source and its properties, as well as accessing (and updating) the *variables* cache associated with the source.

5.1.1.2. Active Change Listeners

The *activeChangeListeners* element defines a list of Active Change Listener implementations that will be instantiated and registered with the active collection.

The fields associated with this component are:

Field	Description
@class	The Java class that provides the listener implementation and is directly or indirectly derived from `org.overlord.rtgov.active.collection.AbstractActiveChan`

The remaining attributes in the example above will be discussed in a subsequent section related to reporting results via JMX notifications.

5.1.1.3. Factory

The *factory* element defines an Active Collection Factory implementation that will be used to create the active collection.

The fields associated with this component are:

Field	Description
@class	The Java class that provides the factory implementation and is directly or indirectly derived from `org.overlord.rtgov.active.collection.ActiveCollectionFa`

The current list of factory implementations are defined below.

Infinispan

The fields associated with the `org.overlord.rtgov.active.collection.infinispan.InfinispanActiveCollectionFactory` component are:

Field	Description
cache	The name of the cache to be presented as an Active Map.
container	The optional JNDI name used to obtain the cache container. If not defined, then the default container will be obtained from the *infinispan.container* property from `overlord-rtgov.properties` file in the `$JBOSS_HOME/standalone/configuration` folder. If the default container is not defined, then cache details will be obtained from the `rtgov_infinispan.xml` file contained within the `overlord-rtgov.war`.

5.1.2. Registering the Source

5.1.2.1. JEE Container

The Active Collection Source is deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-acs.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-acs-loader-jee.jar
|       |-<additional libraries>
```

The `acs.json` file contains the JSON representation of the Active Collection Source configuration.

The `acs-loader-jee.jar` acts as a bootstrapper to load and register the Active Collection Source.

If custom active collection source and/or active change listeners are defined, then the associated classes and resources can be defined in the `WEB-INF/classes` folder or within additional libraries located in the `WEB-INF/lib` folder.

A maven pom.xml that will create this structure is:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.active-queries</groupId>
                        <artifactId>active-collection</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.active-queries</groupId>
                        <artifactId>acs-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                ....
        </dependencies>

</project>
```

If deploying in JBoss Application Server, then the following fragment also needs to be included, to define the dependency on the core Overlord rtgov modules:

```
.....
        <build>
                <finalName>....</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-plugin</artifactId>
                                <configuration>
```

```
                                        <failOnMissingWebXml>false</
failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

 <Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

5.2. Presenting Results from an Event Processor Network

As discussed in the preceding section, an Active Collection Source can be configured to obtain information from an Event Processor Network, which is then placed in the associated Active Collection. This section will explain in more detail how this can be done using the specific Active Collection Source implementation.

```
[
  {
    "@class" :
 "org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "Situations",
    "type" : "List",
    "itemExpiration" : 40000,
    "maxItems" : 0,
    "subject" : "Situations",
    "activeChangeListeners" : [ {
        "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
        "objectName" : "overlord.rtgov:name=Situations",
        "descriptionScript" : "SituationDescription.mvel",
        "insertTypeScript" : "SituationType.mvel"
    } ]
  }
]
```

This configuration shows an example of an Active Collection Source using the `org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource` implementation. The additional fields associated with this implementation are:

Field	Description
subject	The EPN subject upon which the information has been published.

An example Event Processor Network configuration that will publish information on the subject (e.g. *Situations*) specified in the Active Collection Source configuration above is:

```
{
  "name" : "SLAMonitorEPN",
  "subscriptions" : [ {
    "nodeName" : "SLAViolations",
    "subject" : "ServiceResponseTimes"
  } ],
  "nodes" : [
    {
      "name" : "SLAViolations",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.drools.DroolsEventProcessor",
        "ruleName" : "SLAViolation"
      },
      "predicate" : null,
      "notifications" : [ {
         "type" : "Processed",
         "subject" : "SituationsProcessed"
      },{
         "type" : "Results",
         "subject" : "Situations"
      } ]
    }
  ],
  "version" : "1"
}
```

5.3. Publishing Active Collection Contents as JMX Notifications

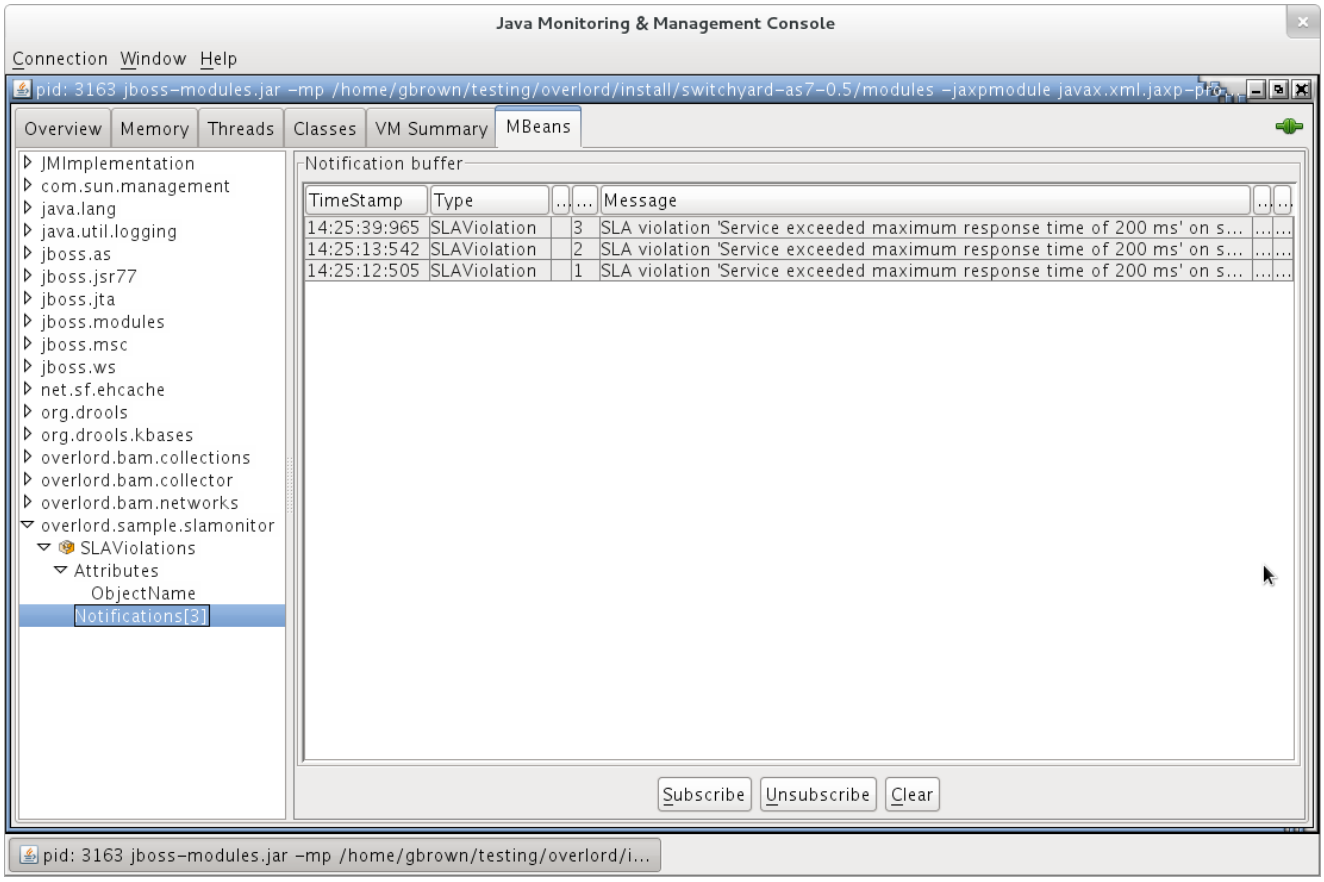
```
[
  .....
 {
   .....
   "activeChangeListeners" : [ {
     "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
     "objectName" : "overlord.sample.slamonitor:name=SLAViolations",
     "insertType" : "SLAViolation"
   } ],
   .....
 }
```

```
]
```

This configuration shows the use of the JMXNotifier active change listener implementation. This implementation has the following additional fields:

Field	Description
objectName	The MBean (JMX) object name to be used to report the notification.
descriptionScript	The MVEL script that can be used to derive the *description* field on the notification. If not defined, then the information's *toString()* value will be used.
insertType	The *type* field for the notification when performing an insert.
insertTypeScript	An optional MVEL script that can be used to derive the *type* field for an insert.
updateType	The optional *type* field for the notification when performing an update.
updateTypeScript	An optional MVEL script that can be used to derive the *type* field for an update.
removeType	The optional *type* field for the notification when performing a removal.
removeTypeScript	An optional MVEL script that can be used to derive the *type* field for a remove.

The following JConsole snapshot shows this JMXNotifier in action, reporting SLA violations from the associated active collection:



5.4. Querying Active Collections via REST

The Active Collections configured within the runtime governance server can be accessed via a REST service, by POSTing a query specification to the URL: `<host>/overlord-rtgov/acm/query`

The Query Specification is comprised of the following information:

Attribute	Description
collection	The active collection name.
predicate	Optional. If defined with the parent name, then can be used to derive a child collection that filters its parent's content (and notifications) based on the predicate.
parent	Optional. If deriving a child collection, this field defines the parent active collection from which it will be derived.
maxItems	Defines the maximum number of items that should be returned in the result, or 0 if unrestricted.

Attribute	Description
truncate	If a maximum number of items is specified, then this field can be used to indicate whether the ***Start*** or ***End*** of the collection should be truncated.
style	Allows control over how the results are returned. The value ***Normal*** means as it appears in the collection. The value ***Reversed*** means the order of the contents should be reversed.

The collection field defines the name of the collection - either an existing collection name, or if defining the *predicate* and *parent* fields, then this field defines the name of the derived collection to be created.

The predicate field refers to a component that implements a predicate interface - the implementation is defined based on the *type* field. Currently only a MVEL based implementation exists, with a single field *expression* defining the predicate as a string.

For example,

```
{
    "parent" : "ServiceResponseTime",
    "maxItems" : 5000,
    "collection" : "OrderService",
    "predicate" : {
        "type" : "MVEL",
        "expression" : "serviceType == \"{urn:switchyard-quickstart-
demo:orders:0.1.0}OrderService\" && operation == \"submitOrder\""
    },
    "truncate" : "End",
    "style" : "Reversed"
}
```

If the Active Collection Manager (ACM) does not have a collection named *OrderService*, then it will use the supplied defaults to create the derived collection. If the collection already exists, then the contents will simply be returned, allowing multiple users to share the same collection.

5.5. Pre-Defined Active Collections

This section describes the list of Active Collections that are provided "out of the box".

5.5.1. ServiceResponseTimes

This active collection is a `list` of `org.overlord.rtgov.analytics.service.ResponseTime` objects.

The response times represent an aggregation of the metrics for a particular service, operation and response/fault, over a configured period. The ResponseTime object has the following fields:

- service type
- operation
- fault - the optional fault name
- timestamp
- average - average response time (where response time represents multiple invocations)
- min - minimum response time (where response time represents multiple invocations)
- max - maximum response time (where response time represents multiple invocations)

5.5.2. Situations

This active collection is a `list` of Situation objects.

The Situation object represents a *situation of interest* that has been detected within the Event Processor Network, and needs to be highlighted to end users. The Situation object has the following fields:

- type - the type of situation
- subject - the identity for the entity related to the situation (e.g. service)
- description - a free format textual description of the situation
- timestamp - when the situation occurred
- severity - "Low", "Medium", "High" or "Critical"
- activity references - optional references to activity events that resulted in the situation

This active collection configuration also publishes it contents via a JMX notifier, based on the following configuration details:

```
[
  {
    ........
  },{
    "@class" :
 "org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "Situations",
    "type" : "List",
    "itemExpiration" : 40000,
    "maxItems" : 0,
```

```
    "subject" : "Situations",
    "activeChangeListeners" : [ {
        "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
        "objectName" : "overlord.rtgov:name=Situations",
        "descriptionScript" : "SituationDescription.mvel",
        "insertTypeScript" : "SituationType.mvel"
        } ]
  }
]
```

5.5.3. ServiceDefinitions

This active collection is a `map` of Service Type name to Service Definition. The Service Definition defines:

- the name of the service type (which is also the key for the map),
- the operations it provides (including request, response and fault message types)
- the operations it consumes (including request, response and fault message types)
- the metrics concerning the operations provided and consumed

An example of a service definition, represented in JSON is:

```
{
    "serviceType":"{http://www.jboss.org/examples}OrderService",
    "operations":[{
        "name":"buy",
        "metrics":{
            "count":30,
            "average":1666,
            "min":500,
            "max":2500
        },
        "requestResponse":{
            "metrics":{
                "count":10,
                "average":1000,
                "min":500,
                "max":1500
            },
            "invocations":[{
                "serviceType":"{http://www.jboss.org/
examples}CreditAgencyService",
                "metrics":{
                    "count":10,
                    "average":500,
                    "min":250,
```

```
                    "max":750
                },
                "operation":"checkCredit"
            }]
        },
        "requestFaults":[{
            "fault":"UnknownCustomer",
            "metrics":{
                "count":20,
                "average":2000,
                "min":1500,
                "max":2500
             }
        }]
    }],
    "metrics":{
        "count":30,
        "average":1666,
        "min":500,
        "max":2500
    }
}
```


The list of service definitions returned from this active collection, and the information they represent (e.g. consumed services), represents a near term view of the service activity based on the configuration details defined in the collection's active collection source. Therefore, if (for example) a service has not invoked one of its consumed services within the time period of interest, then its details will not show in the service definition.

This information is simply intended to show the service activity that has occurred in the recent history, as a means of monitoring the real-time situation to deal with emerging problems.

The duration over which the information is retained is determined by two properties in the ServiceDefinitions active collection source configuration - the "scheduledInterval" (in milliseconds) which dictates how often a snapshot of the current service definition information is stored, and the "maxSnapshots" property which defines the maximum number of snapshots that should be used. So the duration of information retained can be calculated as the scheduled interval multiplied by the maximum number of snapshots.

5.5.4. Principals

This active collection is a `map` of Principal name to a map of named properties. This information is used to convey details captured (or derived) regarding a *principal*. A principal can represent a user, group or organization.

Chapter 6. Available Services

This section describes the "out of the box" additional services that are provided.

6.1. Call Trace

The "Call Trace" service is used to return a tree structure tracing the path of a business transaction (as a call/invocation stack) through a Service Oriented Architecture.

The URL for the service's REST GET request is: `<host>/overlord-rtgov/call/trace/instance?identifier=<value>`

This service has the following query parameters:

Parameter	Description
identifier	This mandatory parameter uniquely identifies the activities associated with the business transaction.

6.2. Service Dependency

The "Service Dependency" service is used to return a service dependency graph as a SVG image. The graph represents the invocation and usage links between services (and their operations), and provides a color-coded indication of areas that require attention. Where *situations* have been detected against services or their operations, this will be flagged on the service dependency graph with an appropriate colour reflecting their severity.

The URL for the service's REST GET request is: `<host>/overlord-rtgov/service/dependency/overview?width=<value>`

This service has the following query parameters:

Parameter	Description
width	Represents the optional image width. If the width is below a certain threshold, then a summary version of the dependency graph will be provided without text or tooltips (used to display metrics).

6.2.1. How to customize the color coding

The colors used for the graph nodes and links can be customized by editing a MVEL script.

The script is called `ColorSelector.mvel` and is located within the `/WEB-INF/classes` folder of the `overlord-rtgov-services.war` archive. This file can be edited and updated within this war, and the war deployed to cause the changes to take affect.

An example of the contents of this script is:

```
String color="#00FF00";
double gap=metric.getMax()-metric.getMin();

if (gap > 0) {
    double mid=metric.getAverage()-metric.getMin();

    double ratio=mid/gap;

    if (ratio > 0.95) {
        color = "#FF0000";
    } else if (ratio > 0.9) {
        color = "#FF3300";
    } else if (ratio > 0.85) {
        color = "#FF5930";
    } else if (ratio > 0.8) {
        color = "#FF6A45";
    } else if (ratio > 0.75) {
        color = "#FF9479";
    } else if (ratio > 0.7) {
        color = "#FF9900";
    }
}

return (color);
```

The script takes two variables:

Variable	Description
metric	The metric to be evaluated.
component	The service definition component associated with the metric. This variable is not used within the example script above.

The script is then responsible for returning the color code.

Chapter 7. Managing The Infrastructure

7.1. Managing the Activity Collector

The Activity Collector mechanism is responsible for collecting activity event information from within a particular execution environment and reporting it as efficiently as possible to the Activity Server.

This section explains how different Activity Collector implementations may be administered.

7.1.1. Batched Activity Collector

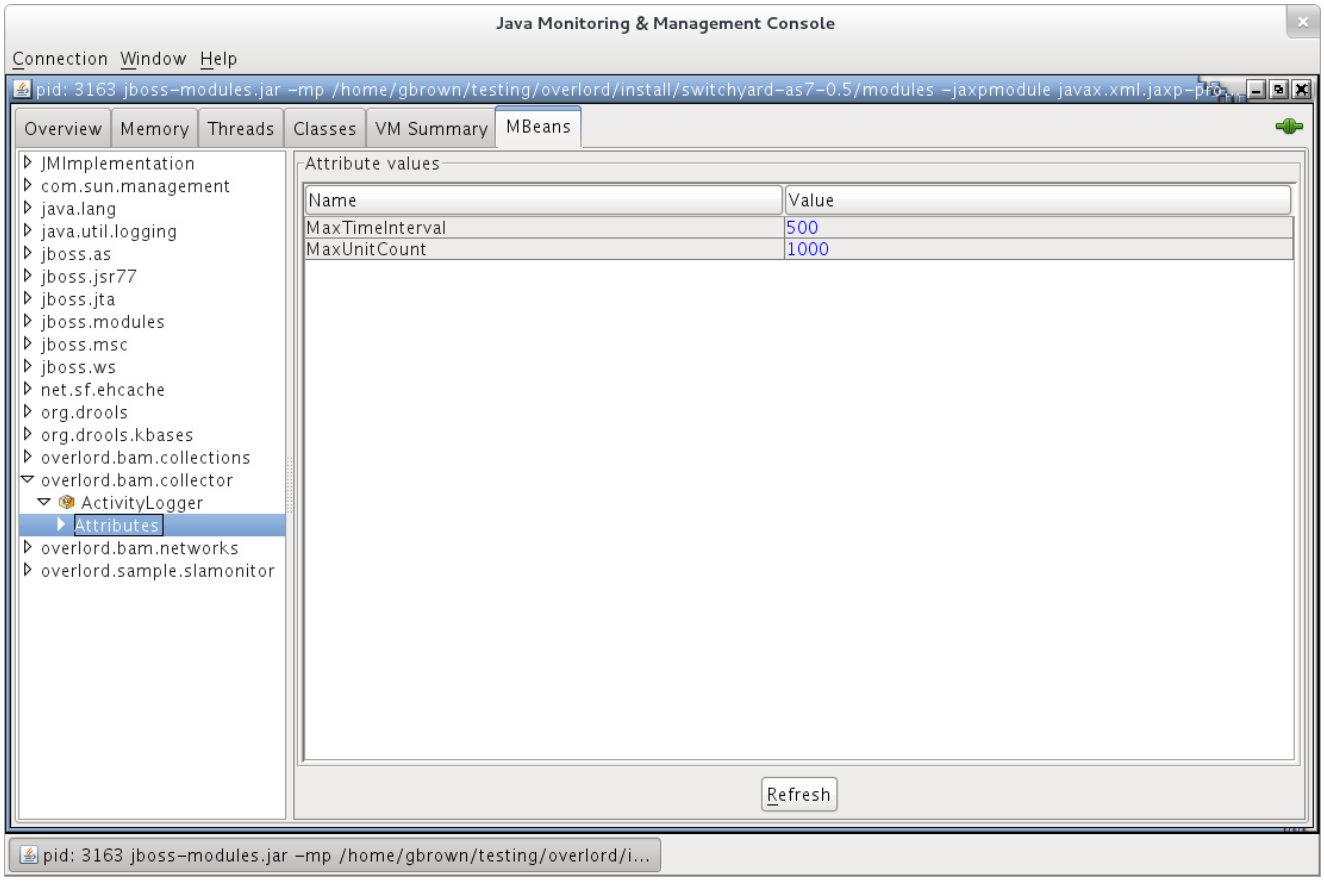
This implementation of the activity collector uses a batching capability to enable the information to be sent to the Activity Server as efficiently as possible.

This mechanism has two configuration properties that can be set on the Activity Unit Logger component:

Property	Description
MaxUnitCount	The maximum number of activity units that should be batched before sending the group to the Activity Server.
MaxTimeInterval	The maximum amount of time (in milliseconds) before sending the batch of events to the server.

The maximum number of items takes precedence, so if it is reached before the defined interval, then the events will be sent to the server.

If the collector is running within a JEE environment, then these properties can be set via a JMX, e.g. using the JConsole:



7.2. Managing the Event Processor Networks

There are two aspects to managing the Event Processor Network mechanism, the *manager* component and the networks themselves. This section will outline the management capabilities associated with both.

7.2.1. Event Processor Network Manager

The Event Processor Network Manager is the component responsible for registering and initializing the Event Processor Networks within a containing environment.

If supported, the manager's attributes and notifications can be exposed via JMX. Currently the attributes that are available:

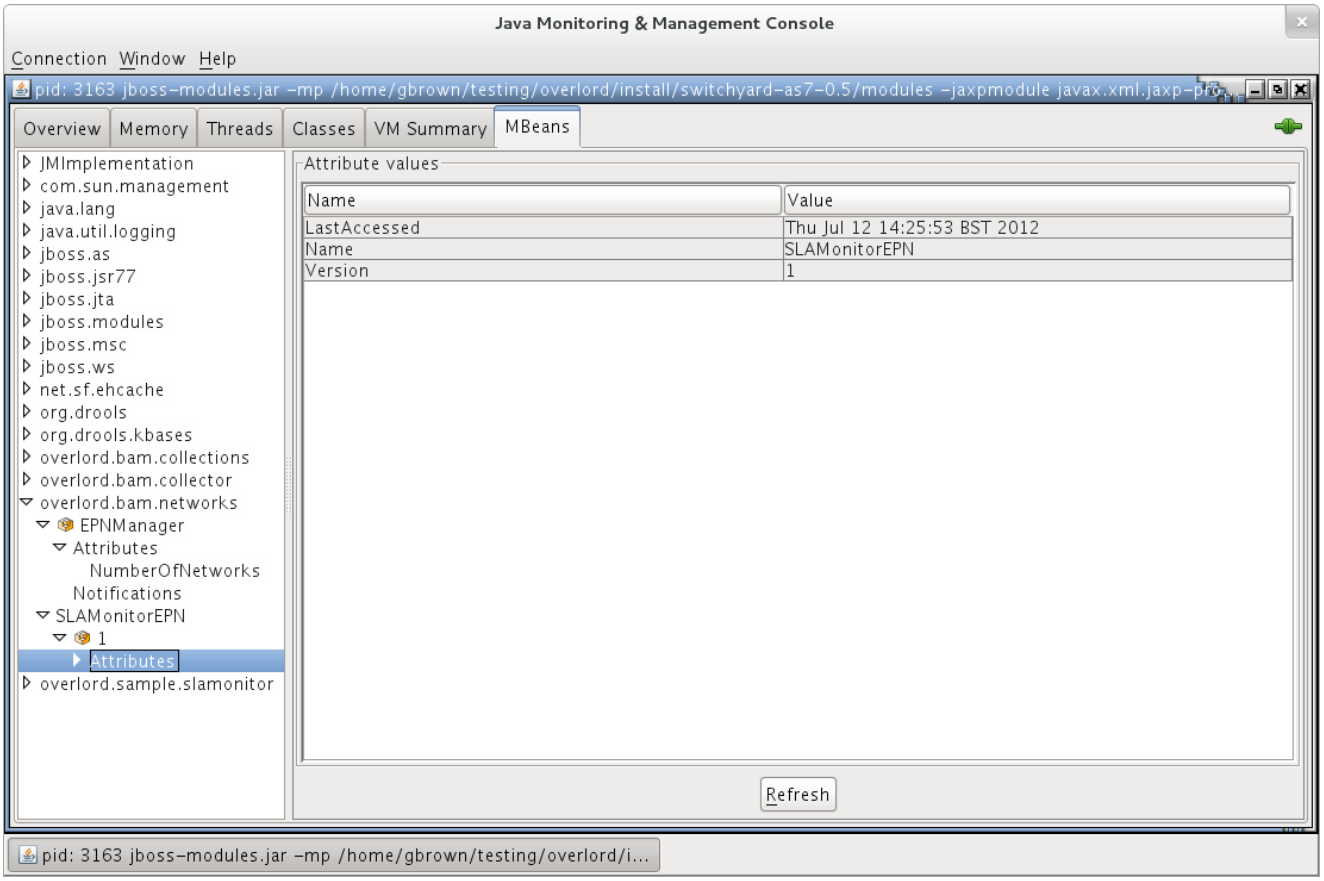
Attribute	Description
NumberOfNetworks	This attribute defines the number of networks registered in the manager.

7.2.2. Event Processor Networks

When a network is registered, if within a JEE environment, it will also be registered as a managed bean, and therefore available via JMX. Each network provides the following attributes:

Attribute	Description
LastAccessed	When the network was last used to process an event. This can be used to determine when it is safe to remove/unregister a network.
Name	The name of the network.
Version	The version of the network.

For example, using the JConsole:



7.3. Managing the Active Collections

There are two aspects to managing the Active Collections mechanism, the *manager* component and the collections themselves. This section will outline the management capabilities associated with both.

7.3.1. Active Collection Manager

The Active Collection Manager is the component responsible for registering and initializing the Active Collection Sources within a containing environment.

If supported, the manager's attributes and notifications can be exposed via JMX. Currently the attributes that are available:

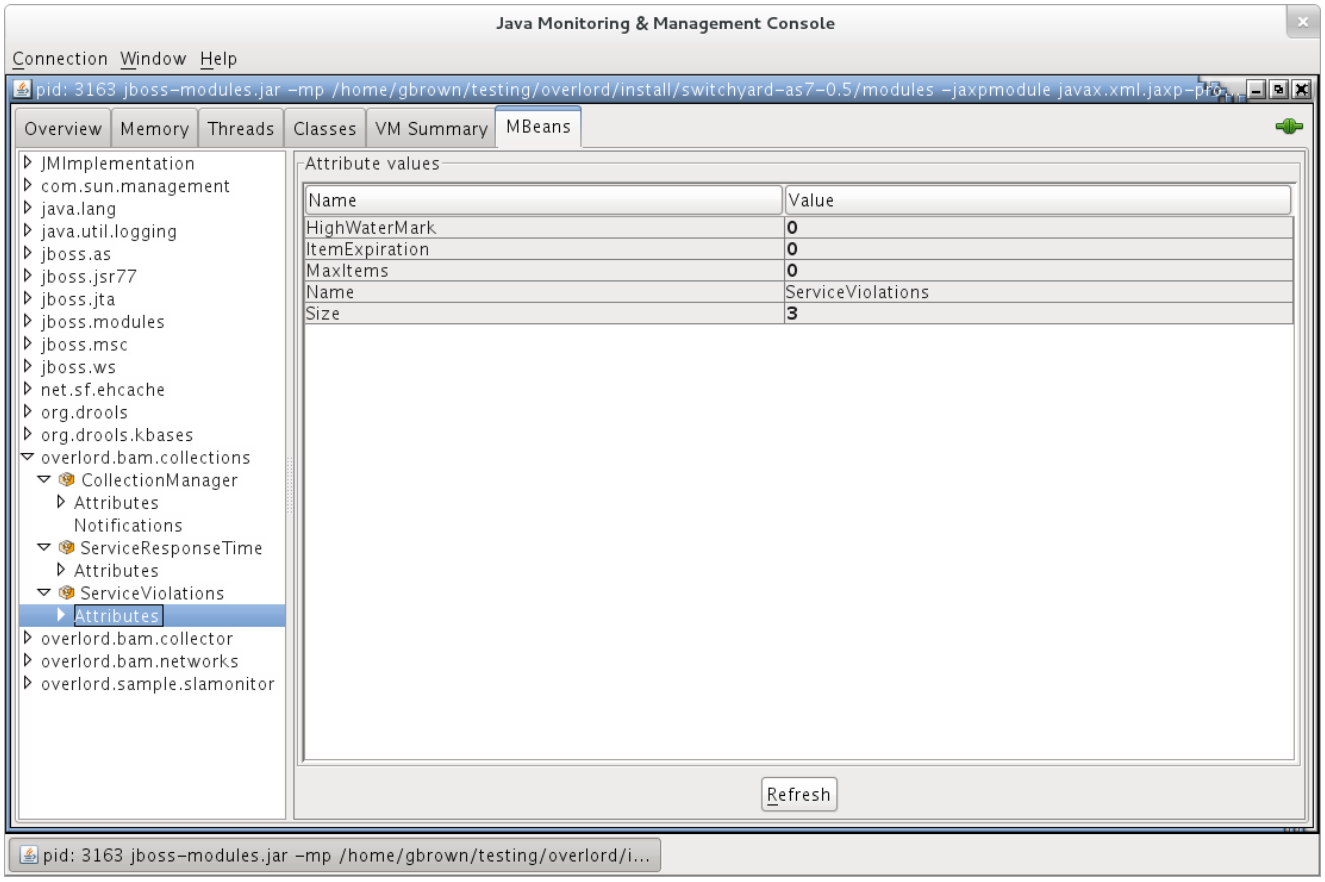
Attribute	Description
HouseKeepingInterval	The number of milliseconds between each house keeping cycle. The house keeping refers to removing items from collections if they are either expired, or the maximum number of elements in the collection has been reached.

7.3.2. Active Collections

When a source is registered resulting in an Active Collection being created, if within a JEE environment, the Active Collection will also be registered as a managed bean, and therefore available via JMX. Each collection provides the following attributes:

Attribute	Description
HighWaterMark	If the number of items in the collection reaches this value, then a warning will be issued. If zero, then does not apply.
ItemExpiration	The number of milliseconds before an item in the collection should be removed. If zero, then does not apply.
MaxItems	The maximum number of items that should be in the collection. If zero, then does not apply.
Name	The name of the Active Collection.
Size	The number of items in the collection.

For example, using the JConsole:


Java Monitoring & Management Console
Connection Window Help
pid: 3163 jboss-modules.jar -mp /home/gbrown/testing/overlord/install/switchyard-as7-0.5/modules -jaxpmodule javax.xml.jaxp-pro
Overview
Memory
Threads
Classes
VM Summary
MBeans
JMImplementation
com.sun.management
java.lang
java.util.logging
jboss.as
jboss.jsr77
jboss.jta
jboss.modules
jboss.msc
jboss.ws
net.sf.ehcache
org.drools
org.drools.kbases
overlord.bam.collections
CollectionManager
Attributes
Notifications
ServiceResponseTime
Attributes
ServiceViolations
Attributes
overlord.bam.collector
overlord.bam.networks
overlord.sample.slamonitor
Attribute values
Name
Value
HighWaterMark
0
ItemExpiration
0
MaxItems
0
Name
ServiceViolations
Size
3
Refresh
pid: 3163 jboss-modules.jar -mp /home/gbrown/testing/overlord/i...