Runtime Governance: User Guide

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

2.1. Setup Target Environment

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

2.1.1. JBoss EAP or Wildfly

- Download the App Server: *JBoss EAP* [http://www.jboss.org/jbossas/downloads/] distribution (version 6.1 and 6.3 are currently supported) or *Wildfly* [http://wildfly.org/downloads/] distribution (version 8.1), and unpack it in a suitable location.
- If using rtgov with switchyard, then download *SwitchYard* [http://www.jboss.org/switchyard/downloads] (version 2.0.0.Final or higher) and install it into the JBoss EAP/Wildfly environment. We recommend using the switchyard installer, which can be unpacked in a temporary location, and run *ant* in the root folder to be prompted for the location of the JBoss EAP or Wildfly environment.

Note

If switchyard is not installed, then you won't be able to use the quickstarts, which are based around providing runtime governance for a switchyard application.

- Download the latest release of RTGov from the *Overlord website* [http://www.projectoverlord.io], choosing the appropriate distribution for the target container and installation type (e.g. client only or all). Unpack the distribution into a suitable location.

Note: The difference between the installation types is,

Value	Description
client	This will result in only the activity collector functionality being installed, using a RESTful client to communicate with a remote Runtime Governance server.
all	This will result in the full server configuration being installed into the server, including activity collector (for obtaining activities generated within that server), activity server (for receiving activity information whether from a remote client or internal activity collector), event processor network (to analyse the events), active collections (to maintain result information) and a collection

Value	Description
	of REST services to support remote access to the information. This is the default value.

- The final step is to perform the installation of Overlord Runtime Governance. To do the installation, use the following command from the root folder of the installation:

```
./install.sh [ -Dpath=<location> ]
```

The *<location>* represents the folder where the JBoss EAP or Wildfly environment exists. If the *<location>* is not explicitly provided on the command line, then the user will be prompted for the information.

To uninstall, simply perform the following command in the root folder of the installation:

```
./uninstall.sh [ -Dpath=<location> ]
```

To start the server, go to the EAP/Wildfly `bin` folder and run:

```
./standalone.sh -c standalone-full.xml
```

The final step is to configure KeyCloak with the Governance realm. This is achieved by following these steps:

- Enter the URL *http://localhost:8080/auth/admin/master/console/#/create/realm* into your browser
- If first time using KeyCloak on the server, then enter the username *admin* and password 'admin. You will then be prompted to enter a new password (twice) for the admin user.
- When the create realm page is displayed, it will offer the ability to upload a realm definition. Select the button and when a file dialog appears, navigate to the *dist* folder within the RTGov distribution and select the *governance-realm.json* file.

You can also import this governance realm by providing the file as an option when starting the server, e.g.

```
./standalone.sh -c standalone-full.xml -Dkeycloak.import=<path-to-
distribution>/dist/governance-realm.json
```

- Once the realm has been uploaded successfully, you will be able to log in to the RTGov UI (*http://localhost:8080/rtgov-ui*) using the username *admin* and password *admin*

2.2. Further Configuration

The configuration properties for the Runtime Governance capability are found in the *<root>/standalone/configuration/overlord-rtgov.properties* file.

Although there will be some properties that are independent of the installation type, some will be specific and therefore are listed in separate sections below.

2.2.1. Common Properties

The common properties available across all installation types are:

Property	Description
collectionEnabled	This property will determine whether activity information is collected when the server is initially started. This value can be changed at runtime using the ActivityCollector MBean (see the chapter on *Managing the Infrastructure*).
ActivityServerLogger.activityListQueueSize	This property defines the queue size for pending activity lists, that are awaiting being reported to the Activity Server.
ActivityServerLogger.durationBetweenFailureReports	To avoid logs being overlorded with failure reports, failures will only be reported once within the defined time interval (in milliseconds).
ActivityServerLogger.freeActivityListQueueSize	This property defines the queue size to manage free activity lists that can be reused.
ActivityServerLogger.maxThreads	This property is an integer that represents the maximum number of threads that should be used to report activity events to the server (whether remote or embedded).
BatchedActivityUnitLogger.maxTimeInterval	The maximum wait interval (in milliseconds) before sending any held activity units to the Activity Server.
BatchedActivityUnitLogger.maxUnitCount	The maximum number of activity units that should be held before sending as a batch to the Activity Server.

2.2.2. Properties specific to the full installation

Property	Description
ActiveCollectionManager.houseKeepingInterval	Time interval (in milliseconds) between house keeping tasks being invoked.
ActivityStore.class	The class associated with the Activity Store implementation to be used.
Elasticsearch.server	URL to the Elasticsearch server (HTTP port).

Property	Description
infinispan.container	The infinispan container to use.
MVELSeverityAnalyzer.scriptLocation	Optional location of a MVEL script used to determine severity levels for nodes and links within the service overview diagram.
SituationStore.class	The class associated with the Situation Store implementation to be used.

Note

Activity and Situation Store implementation specific properties will be discussed in the database section below.

As part of the full installation, the RTGov UI is also installed, and includes the following additional properties:

Property	Description
SwitchYardServicesProvider.serverURLs	A comma separated list of URLs that can be used to resubmit messages to a SwitchYard server. The URLs are used with a round robin strategy. (The default values is *http://localhost:8080*)
SwitchYardServicesProvider.jmxURL	The URL for the remote MBeanServer connection. If no value is provided, then a local MBeanServer connection will be used.
SwitchYardServicesProvider.jmxUsername	Username for the remote MBeanServer connection.
SwitchYardServicesProvider.jmxPassword	Password for the remote MBeanServer connection.
UI.manualResolution	Resolution state will automatically be changed based on the outcome of a resubmission, but this boolean property determines whether resolution state change buttons should be included on the Situation details page, to allow a user to manually change the state. The default value is "true".

2.2.3. Properties specific to the *client only* installation

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. The relevant properties are:

Property	Description
RESTActivityServer.serverURL	This is the URL of the activity server collecting the activity events.
RESTActivityServer.serverUsername	The username used to access the REST service.
RESTActivityServer.serverPassword	The password used to access the REST service.

2.2.4. Database

This section described the configuration of the supported database options.

2.2.4.1. Elasticsearch

Note

This is the default "out of the box" configuration.

To use Elasticsearch as the Activity and Situation Store implementation, the following property values need to be defined:

```
ActivityStore.class=org.overlord.rtgov.activity.store.elasticsearch.ElasticsearchActivitySto
SituationStore.class=org.overlord.rtgov.analytics.situation.store.elasticsearch.Elasticsearc
```

with the additional support properties:

Property	Description
Elasticsearch.hosts	Either has value "embedded" (the default), or a list of <host>:<port> values representing nodes in the Elasticsearch cluster, the port representing the TCP transport connection.
Elasticsearch.schedule	When using batched mode, the interval (in milliseconds) between updates being sent to the Elasticsearch server.
Elasticsearch.ActivityStore.responseSize	Maximum size for the response (default value 100000).
Elasticsearch.ActivityStore.timeout	"Best effort" timeout value (milliseconds) (default value 10000ms).
Elasticsearch.SituationStore.responseSize	Maximum size for the response (default value 100000).
Elasticsearch.SituationStore.timeout	"Best effort" timeout value (milliseconds) (default value 10000ms).

The following information describes the Elasticsearch clustering options that are supported with RTGov. For more information please see *http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/modules-node.html*

Out of the box, RTGov starts up with an in-VM Elasticsearch node for convenience. Such a setup is not recommended for a production environment for the following reasons:

- Elasticsearch running on the same JVM could result in resource contention, e.g. memory or cpu, which could impact the application performance
- In a clustered or load-balanced environment we would require Elasticsearch to persist the data to the same cluster

RTGov does not attempt to wrap or hide the standard Elasticsearch configurations. If you know how to tweak and tune an Elasticsearch node then these configuration changes can be applied to the appropriate location (dependent upon platform):

Value	Description
EAP or Wildfly	The configuration properties for the Runtime Governance capability are found in the *<root>/standalone/configuration/overlord-rtgov.properties* file.

If you want to learn how to configure and tune Elasticsearch then please reference the Elasticsearch documentation at *http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/setup-configuration.html*

Some of those configuration properties that may need to be changed include:

- *cluster.name*: Cluster name identifies your cluster for auto-discovery. If you're running multiple clusters on the same network, make sure you're using unique names
- *node.name*: Node names are generated dynamically on startup, so you're relieved from configuring them manually. However you can tie a node to a specific name
- *path.data*: Path to directory where to store index data allocated for this node

There are 3 ways Elasticsearch cluster communication can be configured within RTGov:

Local Elasticsearch embedded server

```
node.local=true
```

This configuration does not communicate outside of the VM, only performing discovery of Elasticsearch nodes started on the same same VM.

Client only with no local data

When you start an Elasticsearch client, the most important decision is whether it should hold data or not. In other words, should indices and shards be allocated to it. Many times we would like to have the clients just be clients, without shards being allocated to them. This is simple to configure by setting either:

```
node.data=false
```

and/or

```
node.client=true
```

With this configuration, the client is cluster aware and can route its data to the responsible shards avoiding a double hop.

Clustered client with local data

This is the default "out of the box" configuration for RTGov. This starts a simple Elasticsearch node that can hold data and also join other Elasticsearch nodes in a cluster.

```
node.data=true
node.client=false
node.local=true
```

2.2.4.2. SQL

To use a SQL database as the Activity and Situation Store implementation, the following property values need to be defined:

Property	Value
ActivityStore.class	`org.overlord.rtgov.activity.store.jpa.JPAActivityStore`
SituationStore.class	`org.overlord.rtgov.analytics.situation.store.jpa.JPASit`

with the additional support properties:

Property	Description
JPAActivityStore.jndi.datasource	The JNDI name used to retrieve the datasource.
JPAEventProcessor.jndi.datasource	The JNDI name used to retrieve the datasource.
JPASituationStore.jndi.datasource	The JNDI name used to retrieve the datasource.
JpaStore.jtaPlatform	The JTA platform Java implementation class.

Warning

As of RTGov 2.x, Elasticsearch is the main supported implementation of the Activity and Situation Store.

2.3. Test the installation using the samples

When RTGov has been installed, try out the samples to get an understanding of its capabilities, and check that your environment has been correctly installed/configured.

2.3.1. JBoss EAP

To install the samples into JBoss EAP go to the `samples` folder in the distribution. You will need to install *Apache Maven* [http://maven.apache.org/download.cgi] to be able to use the examples.

The key examples are explained below. Each quickstart also has a readme providing the instructions for use.

2.3.1.1. Order Management

The `samples/ordermgmt` folder contains examples related to an Order Management system implemented using a SwitchYard application.

The `samples/ordermgmt/app` folder contains the switchyard application, with some additional interceptors to execute policies synchronously (see Activity Validators section for more information, and the Synchronous Policy quickstart more a specific example of its use).

The `samples/ordermgmt/epn` folder contains an Event Processor Network (see later section for details) that is used to convert switchyard application exceptions into "Situations", which is a form of alert used by the Runtime Governance platform.

The `samples/ordermgmt/ip` folder contains an Information Processor (see later sectoin for details) that is used to extract additional information from message payloads, that will be useful when analysing the activity events.

2.3.1.2. Policy

The `samples/policy/sync` folder contains a policy that is invoked synchronously - it determines whether a user has invoked the service more than once every two seconds, and if so, blocks the service invocation.

The `samples/policy/async` folder contains a policy for asynchronously calculating the debt associated with a customer, and suspending their account if it goes above a defined level. The suspended status of the customer is checked when they next invoke the service, and the service invocation blocked if they have been suspended.

2.3.1.3. SLA

The `samples/sla/epn` folder contains a policy for determining whether a Service Level Agreement has been violated, and if so, reported as a *Situation*.

The `samples/sla/monitor` folder contains a webapp that directly integrates with the RTGov components.

2.4. JBoss EAP Specific Information

2.4.1. SQL Database

The database is defined by the datasource configuration located here: `$JBOSS_HOME/standalone/deployment/overlord-rtgov/rtgov-ds.xml` as part of the *server* installation type.

The default SQL database is the H2 file based database, and is created during the installation of the *all* type.

Note

The following sections discuss changes to the `standalone-full.xml` configuration file. If using a clustered environment, then these changes should be applied to the `standalone-full-ha.xml` instead.

MySQL

- Create the folder `$JBossAS/modules/mysql/main`.
- Put the MySQL driver jar in the `$JBossAS/modules/mysql/main` folder, e.g. mysql-connector-java-5.1.12.jar.
- Create a module.xml file, within the `$JBossAS/modules/mysql/main` folder, with the contents:

```
<module xmlns="urn:jboss:module:1.1" name="mysql">
   <resources>
     <resource-root path="mysql-connector-java-5.1.12.jar"/>
   </resources>
   <dependencies>
     <module name="javax.api"/>
     <module name="javax.transaction.api"/>
   </dependencies>
</module>
```

- Edit the `$JBossAS/standalone/configuration/standalone-full.xml` file to include the MySQL driver:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
        <datasources>
            .....
            <drivers>
                ...
                <driver name="mysql" module="mysql">
                                                                    <xa-datasource-
class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-class>
                </driver>
            </drivers>
        </datasources>
    </subsystem>
```

- Update the rtgov datasource file, `$JBossAS/standalone/deployments/overlord-rtgov/rtgov-ds.xml`, the contents should be:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
          <datasource   jndi-name="java:jboss/datasource/OverlordRTGov"   pool-
name="OverlordRTGov" enabled="true" use-java-context="true">
        <connection-url>jdbc:mysql://localhost:3306/rtgov</connection-url>
        <driver>mysql</driver>
        <security>
            <user-name>root</user-name>
            <password></password>
        </security>
    </datasource>
</datasources>
```

Postgres

- Create the `$JBossAS/modules/org/postgresql/main` folder.
- Put the postgresql driver jar in the `$JBossAS/modules/org/postgresql/main` folder, e.g. postgresql-9.1-902.jdbc4.jar.
- Create a module.xml file, within the `$JBossAS/modules/org/postgresql/main` folder, with the contents:

```
<module xmlns="urn:jboss:module:1.1" name="org.postgresql">
   <resources>
     <resource-root path="postgresql-9.1-902.jdbc4.jar"/>
   </resources>
   <dependencies>
```

```
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
 </module>
```

- Edit the `$JBossAS/standalone/configuration/standalone-full.xml` file to include the PostgresSQL driver:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
        <datasources>
            .....
            <drivers>
                ...
                <driver name="postgresql" module="org.postgresql">
                    <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-
datasource-class>
                </driver>
            </drivers>
        </datasources>
    </subsystem>
```

- Update the rtgov datasource file, `$JBossAS/standalone/deployments/overlord-rtgov/rtgov-ds.xml`, the contents should be:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
            <datasource  jndi-name="java:jboss/datasource/OverlordRTGov"  pool-
name="OverlordRTGov" enabled="true" use-java-context="true">
        <connection-url>jdbc:postgresql://localhost:5432/rtgov</connection-url>
        <driver>postgresql</driver>
        <security>
            <user-name>....</user-name>
            <password>....</password>
        </security>
    </datasource>
</datasources>
```

2.4.2. Caching

The EPN and Active Collection mechanisms both have the ability to make use of caching provided by infinispan. When running the server in clustered mode (i.e. with `standalone-full-ha.xml`).

First step is to uncomment the *infinispan.container* property in the `overlord-rtgov.properties` file and set it to the JNDI name of the cache container (*java:jboss/infinispan/container/rtgov* in the

example below). This property represents the default cache container to be used by EPN and Active Collection Source configurations that do not explicitly provide a container JNDI name.

The next step is to create the cache container configuration, and the specific caches, under the *infinispan* subsystem in 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 examples:

```
            <cache-container name="rtgov" jndi-name="java:jboss/infinispan/
container/rtgov" start="EAGER">
                <transport lock-timeout="60000"/>
                <replicated-cache name="Principals" mode="SYNC">
                    <locking isolation="REPEATABLE_READ"/>
                    <transaction mode="FULL_XA" locking="PESSIMISTIC"/>
                </replicated-cache>
            </cache-container>
```

Chapter 3. Visualising the Runtime Governance Information

This section describes how to use the Runtime Governance User Interface (UI).

3.1. Accessing the Runtime Governance UI

To access the Runtime Governance UI, after the server has been started, use the url: `<host>/rtgov-ui`

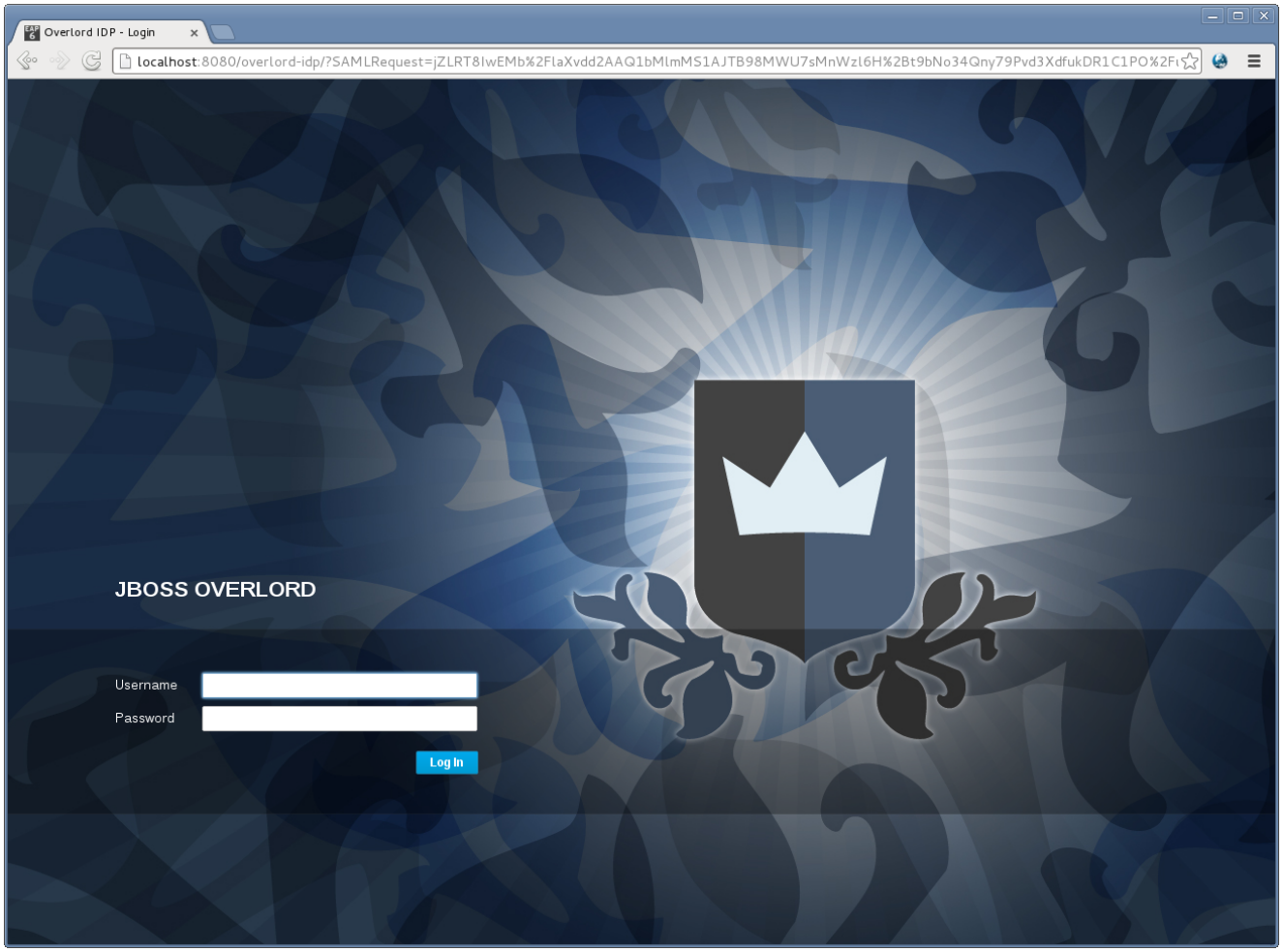

Figure 3.1.

Once displayed, it will request an username and password. When successfully logged in, you will be presented with the top level dashboard:

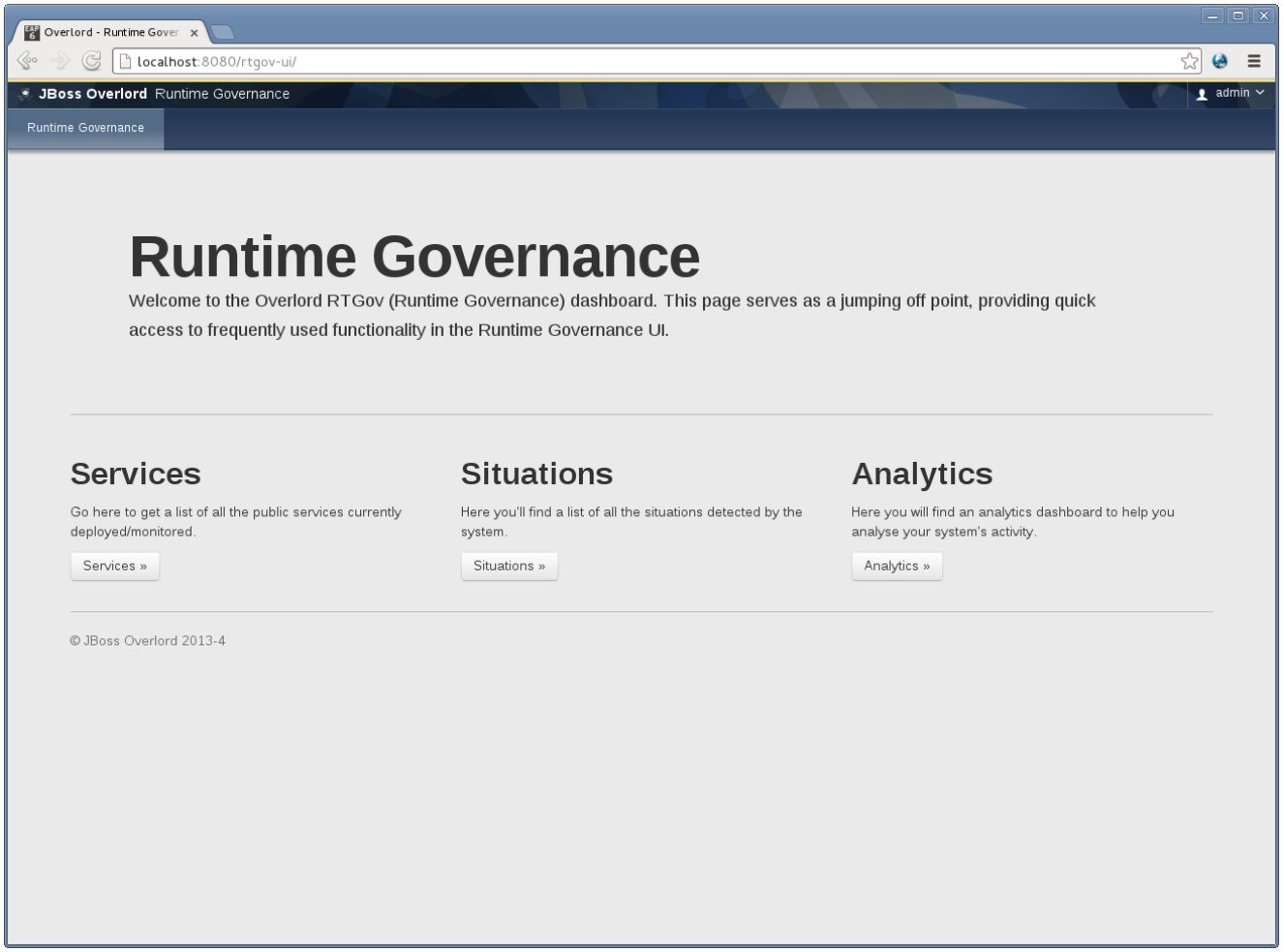

Figure 3.2.

The dashboard provides access to three types of information related to runtime governance. These will be discussed in more detail in the following sections.

3.2. Services

The *Services* page lists the services that have been deployed to a service container (e.g. switchyard) and are being monitored by RTGov.

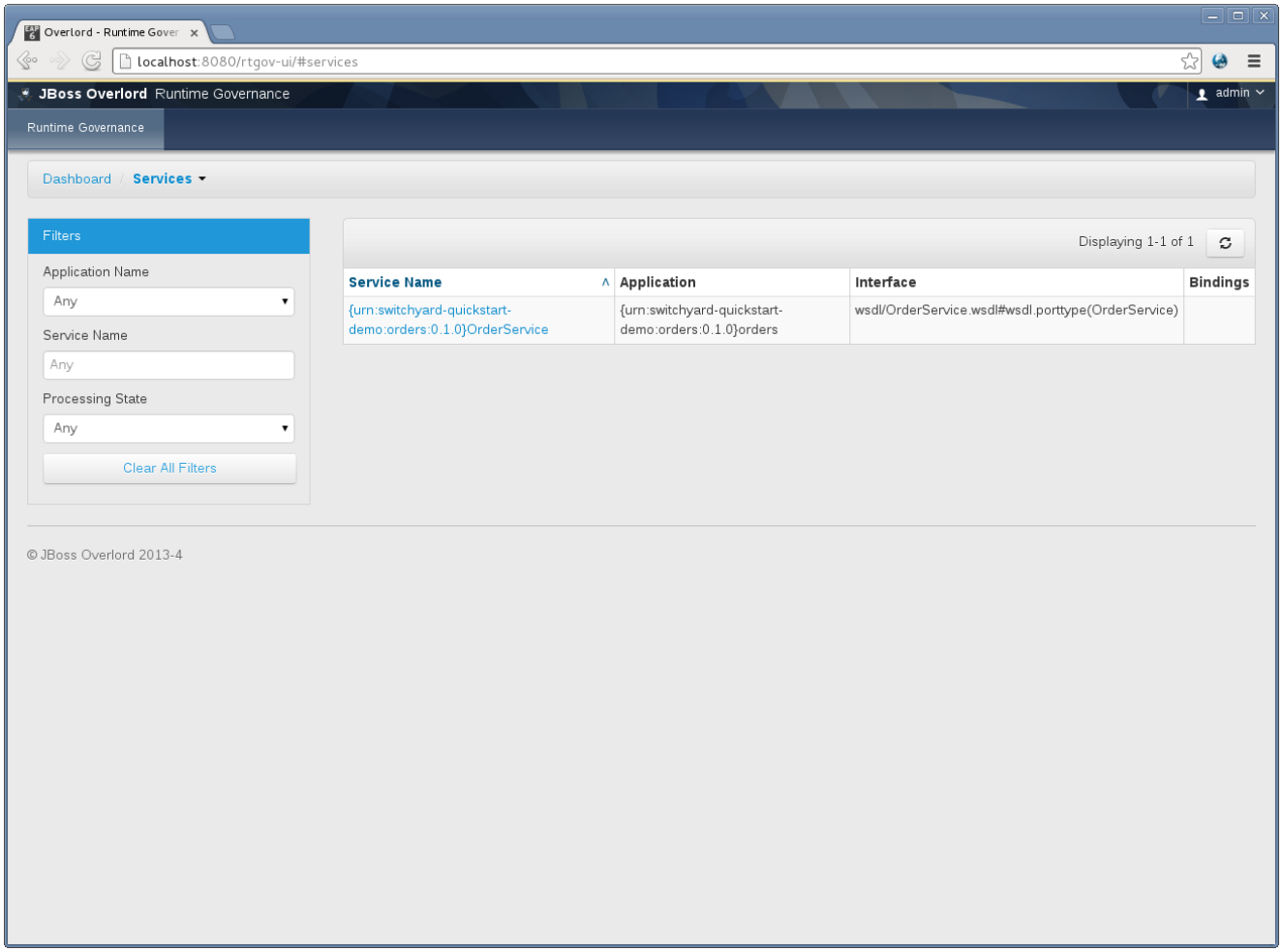

Figure 3.3.

The list shows the service name, optional application in which it is contained, the external interface(s) it implements and finally the bindings through which it can be accessed.

Note

For switchyard services, this list will only contain the public (promoted) services, however activity information will be collected for internal component services as well.

When a service name is selected, it will navigate to the details page:

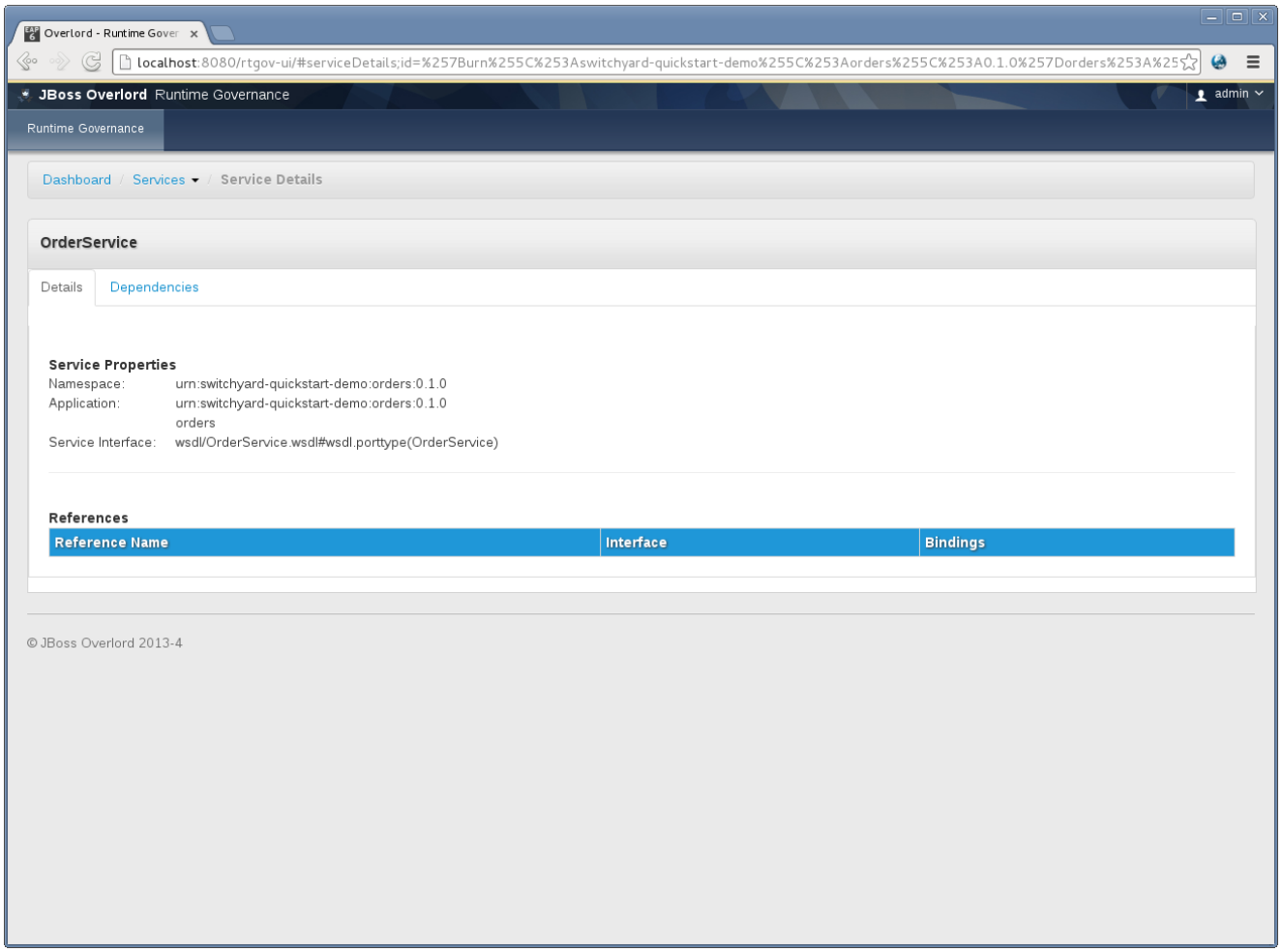

Figure 3.4.

This page shows high level information about the service, and where appropriate, any promoted references it has to other external services.

The *Dependencies* tab can be used to view dynamic dependency information about the service. This information is based on a short term rolling window, and will therefore only show the relationships associated with recent invocations:

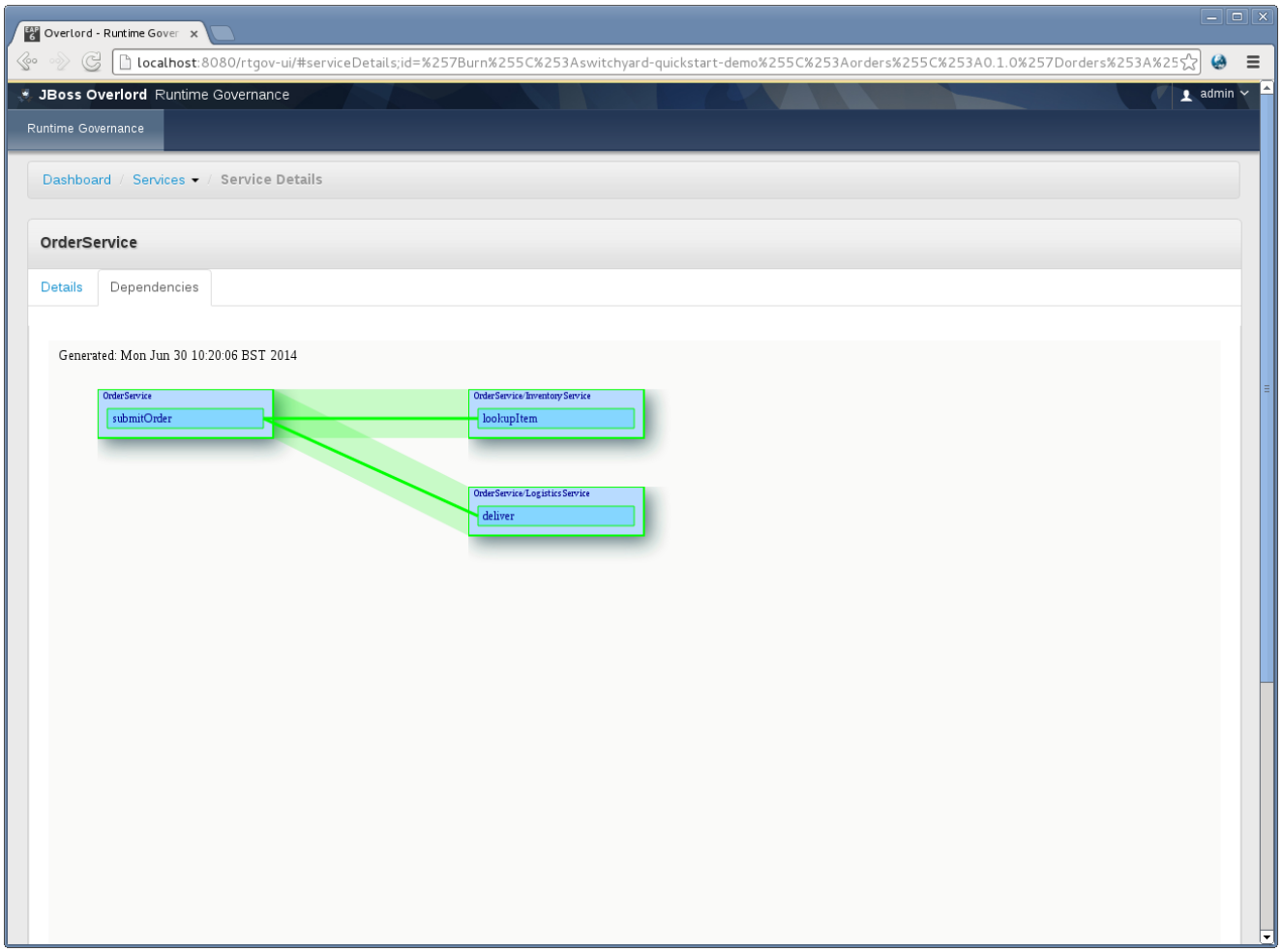

Figure 3.5.

3.3. Situations

3.3.1. Situations List

This section shows the *Situations* that are reported when RTGov policies detect issues that need to be bought to the attention of users.

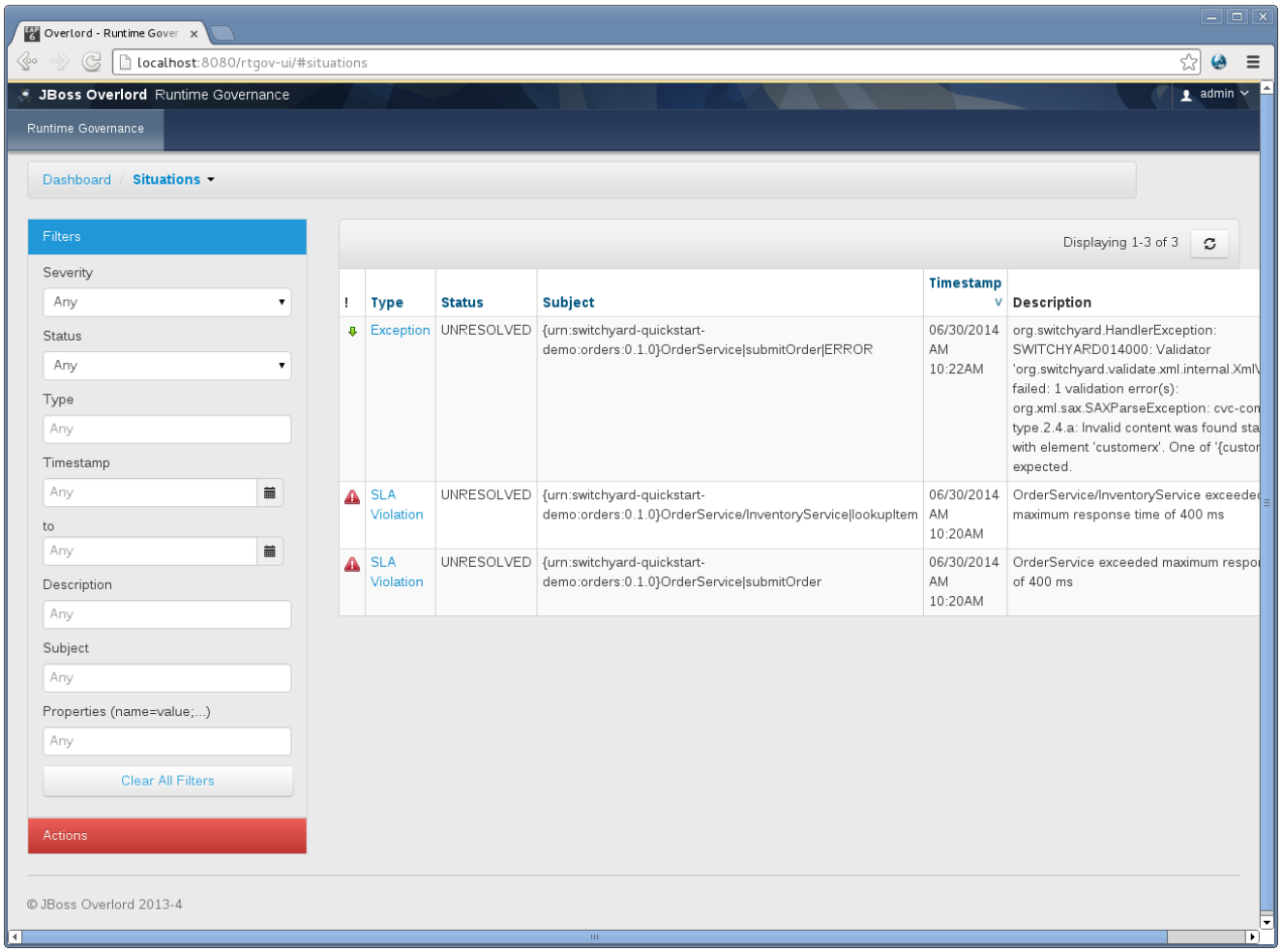

Figure 3.6.

The left hand panel provides a variety of options for filtering the list of situations.

The list contains the following columns:

- Severity - an icon to indicate how severe the situation is.
- Type - identifies the nature of the situation (e.g. SLA Violation, Exception, etc).
- Status - where the situation is in its lifeycycle (see further down for description of the lifecycle). The status *Open* represents all non-resolved states.
- Subject - the subject of the situation, which will generally be a service type and operation, with optional fault name.
- Timestamp - when it occurred.
- Description - further details about the situation.
- Action - show properties for the situation.

At the bottom left is a collapsed region containing controls for performing *bulk actions*.

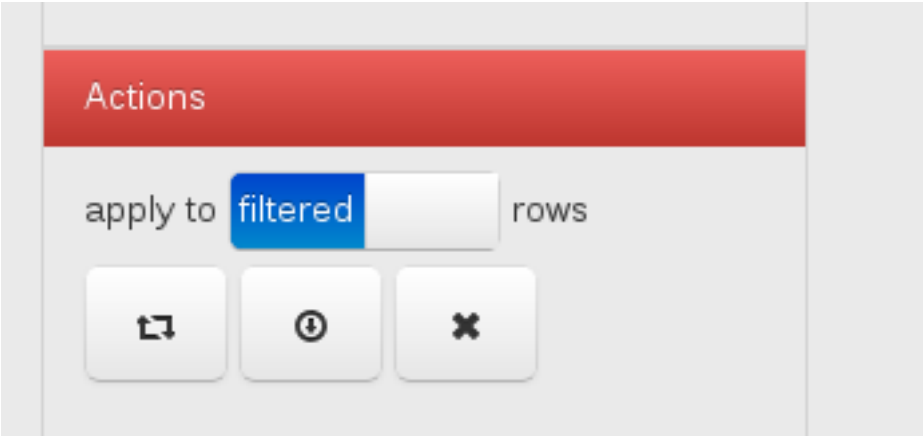

Figure 3.7.

These actions can operate either on the filtered situations (as shown here), or all situations. The actions themselves, from left to right, are *resubmit* (i.e. resubmit an associated message to the target service), *export* and *remove*.

When a new *Situation* occurs, if the user is already viewing the situations page, then a small notification will be displayed in the top right corner:

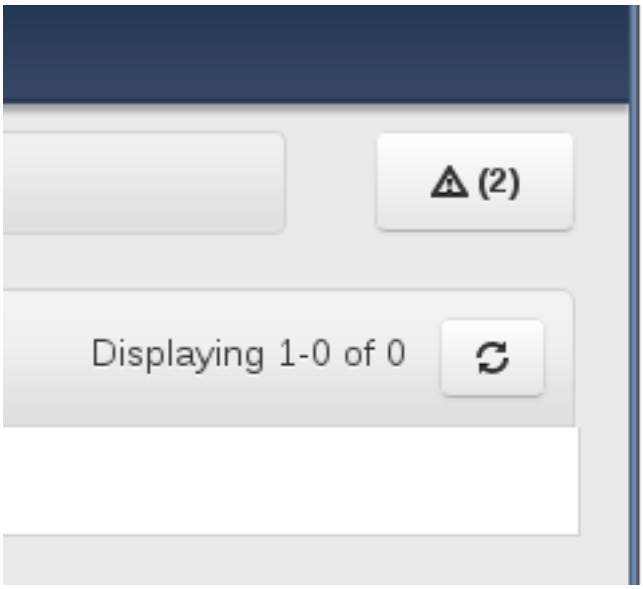

Figure 3.8.

If this notification is expanded, it will list some of the details for the new *Situations*:

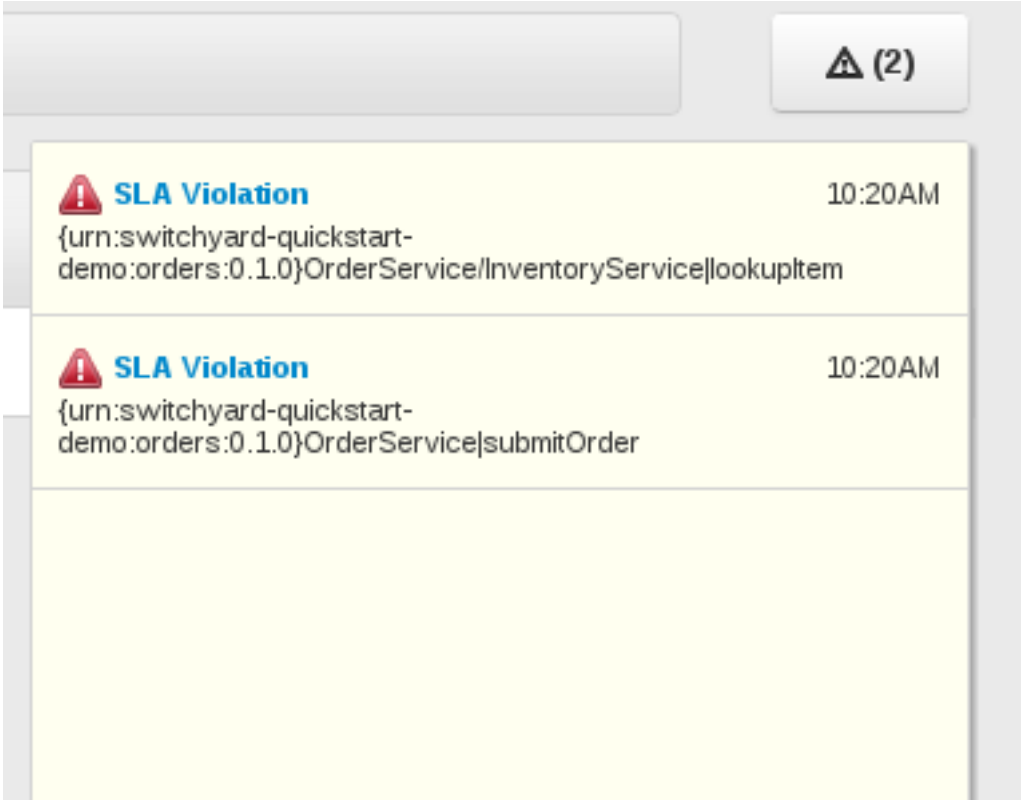

Figure 3.9.

The user can either select one of the entries to navigate to it's details, or alternatively use the refresh button to update the list.

To view the details associated with a *Situation* in the list, select its type field, which will navigate to the details page.

3.3.2. Situation Details

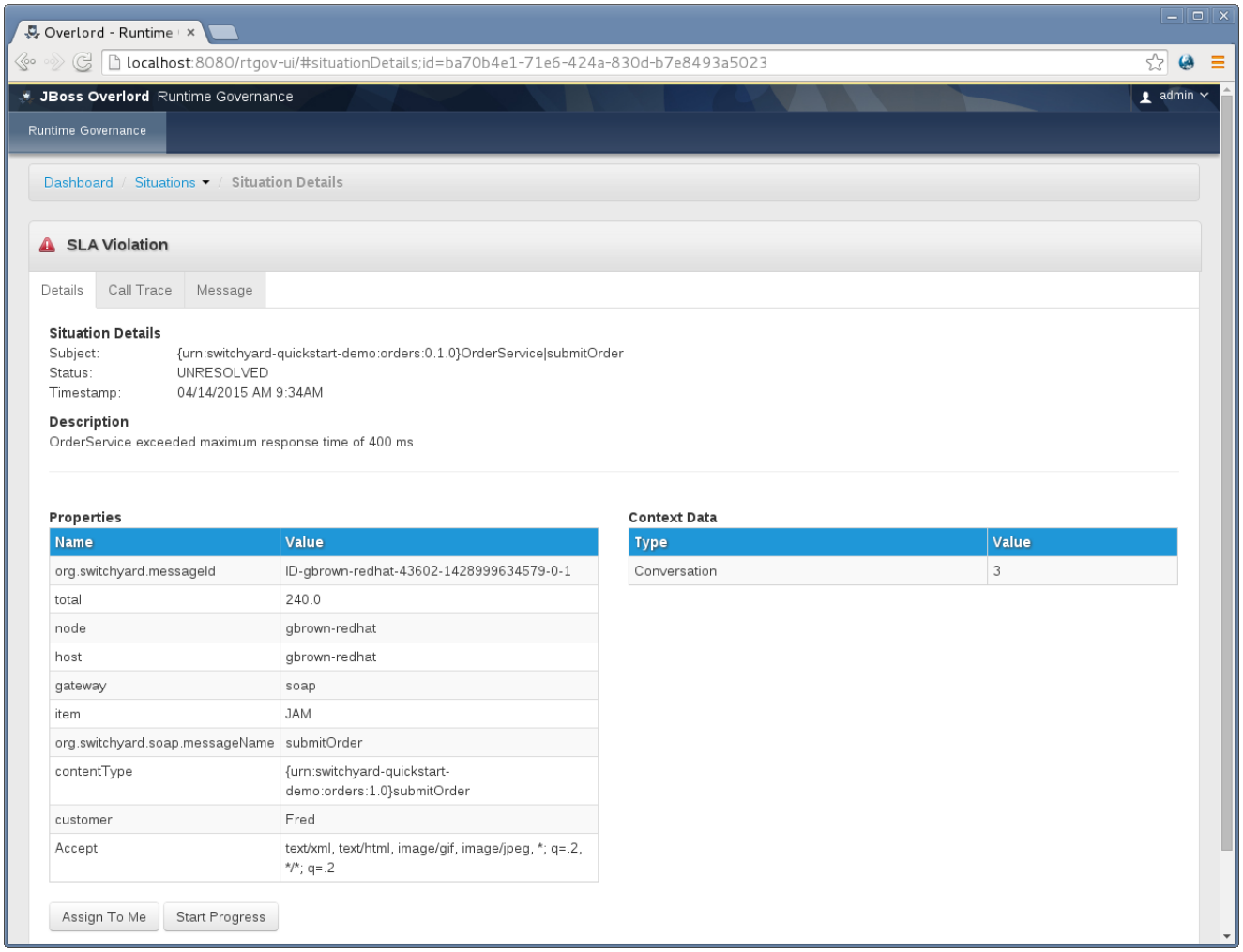

Figure 3.10.

This page shows the details of the situation, including properties and context data.

The *Call Trace* tab shows the call stack associated with the business transaction, if appropriate context information has been recorded with the situation.

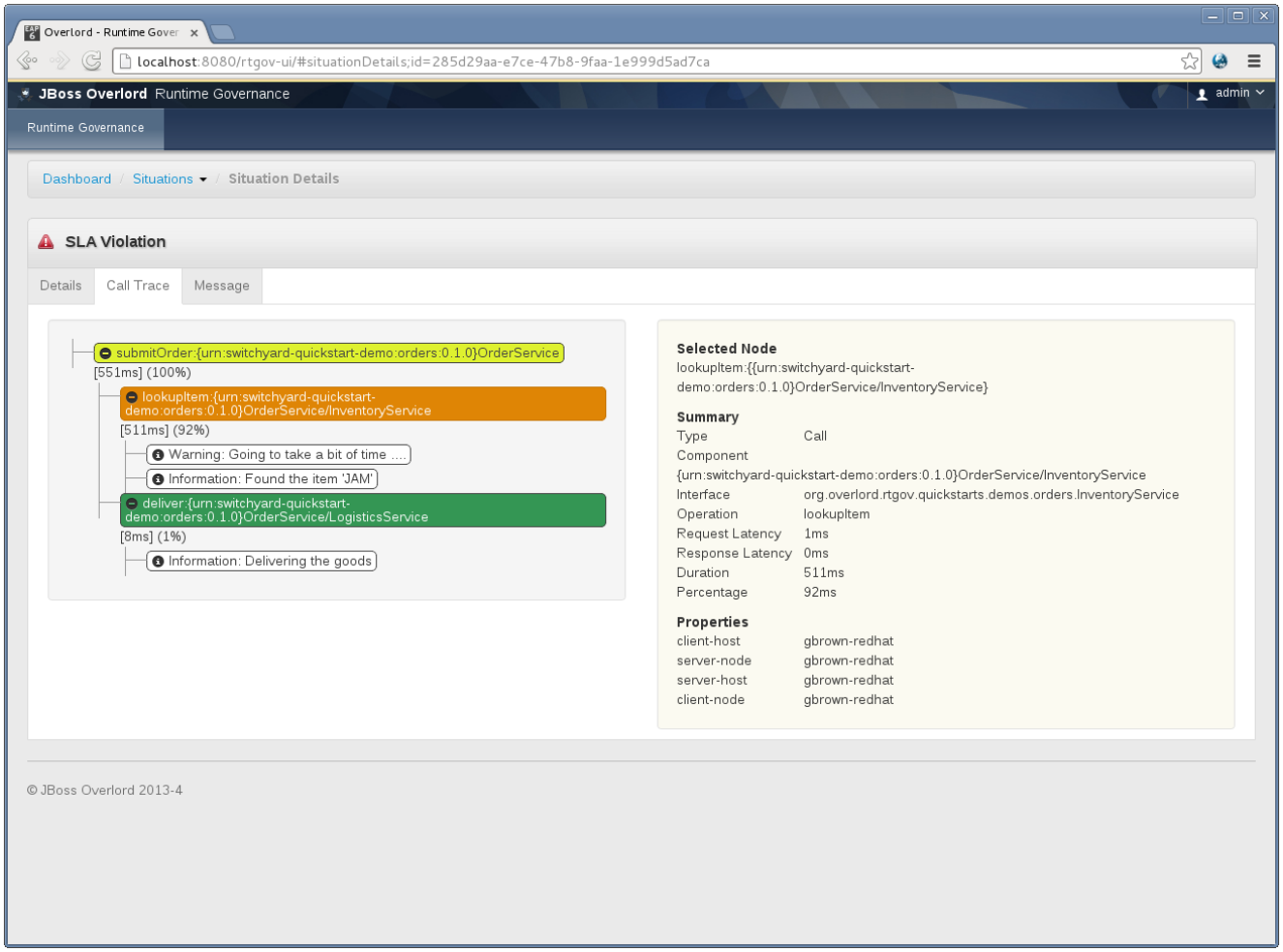

Figure 3.11.

Selecting a node in the call trace displays further details in the right hand panel.

The optional *Message* tab is displayed if the *Situation* has an associated business message.

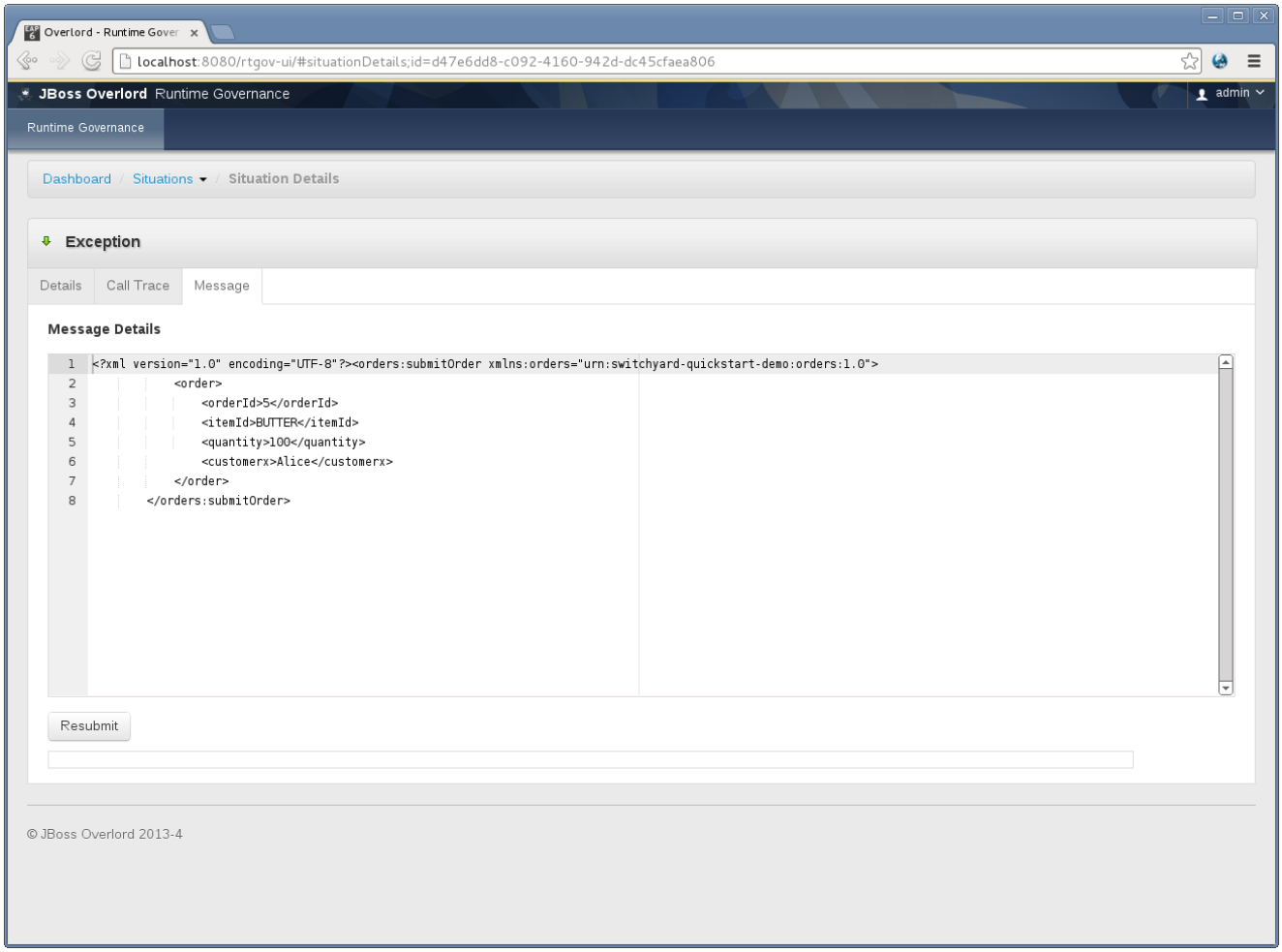

Figure 3.12.

If the service and operation, associated with the situation, supports resubmission of the messages (i.e. if a SwitchYard service, then it would need to have an SCA binding and the operation would need to be one-way), and the *Situation* is not in a RESOLVED state, then the user will be able to edit the message content and press the *Resubmit* button. This will result in resubmission information being displayed:

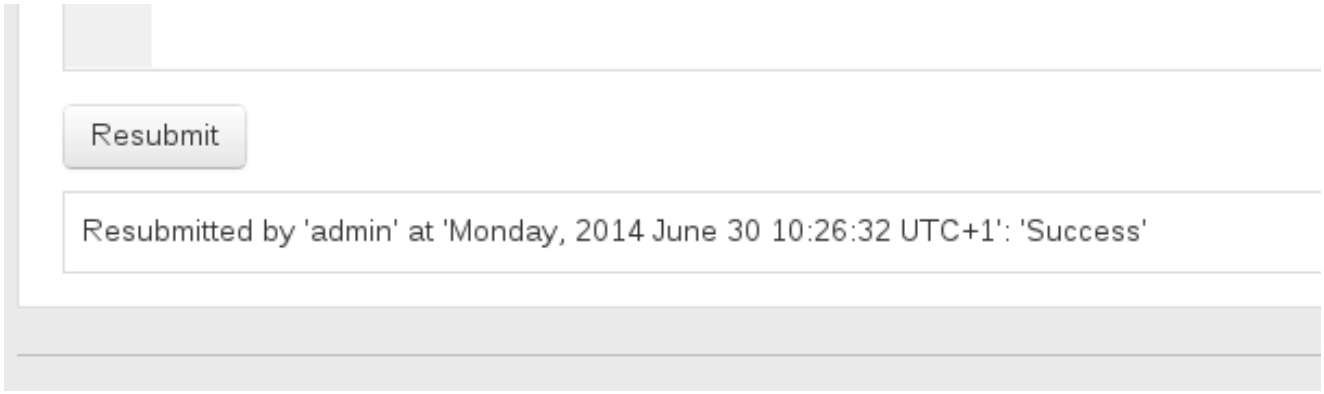

Figure 3.13.

3.3.2.1. Situation Lifecycle

The following diagram shows the lifecycle of a *Situation*. Transitions are controlled by the outcome of a resubmission associated with the situation, or appropriate buttons presented when viewing the situation details (if manual resolution is permitted).

Figure 3.14.

3.3.2.2. Situation Resubmission

When a resubmission is performed, the outcome of the resubmission will be used to determine whether the associated *Situation* can be automatically marked as RESOLVED (if successful) or IN_PROGRESS (if unsuccessful).

Any subsequent situations created as a result of a resubmission failure will be associated with the *Situation* that was resubmitted, effectively creating a tree of Situations (i.e. subsequent resubmission failures will then be defined as further nodes in the tree). However, only the top level *Situation* will be shown within the list of situations.

To indicate which of the situations in the list have associated Situations (due to resubmission failures), a numerical value will be placed in the first column:

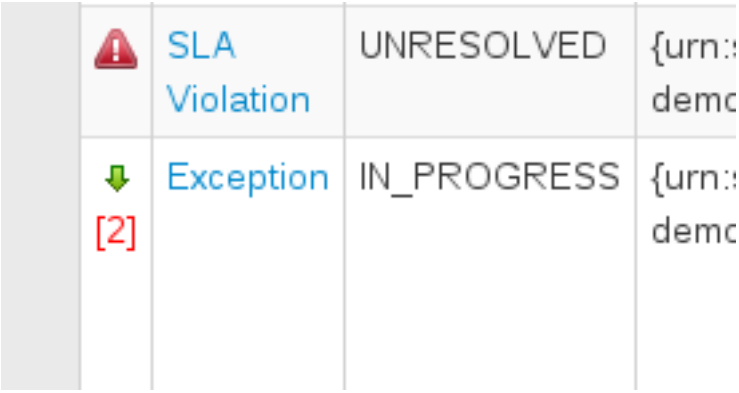

Figure 3.15.

When selecting the details page for such a *Situation*, there will appear a new tab called *Resubmit Failures*, listing all of the situations directly or indirectly contained by the selected *Situation* (i.e. a flattened tree).

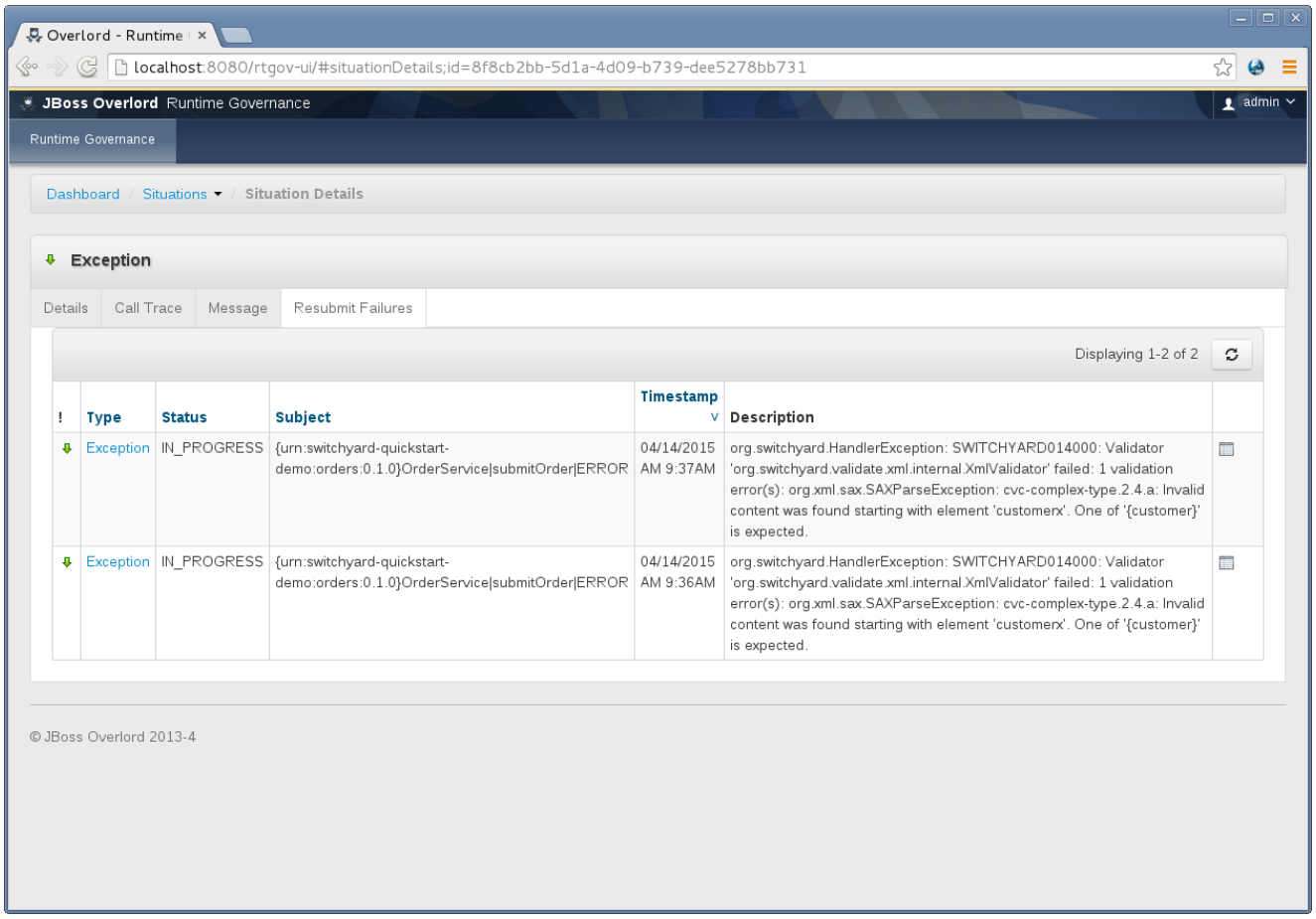

Figure 3.16.

3.4. Analytics

RTGov uses *Elasticsearch* [http://www.elasticsearch.org/] to store the activity information, and *Kibana* [http://www.elasticsearch.org/overview/kibana/] to provide a dashboard for analysing that information.

3.4.1. Dashboard

The "out of the box" dashboard layout presents the following information:

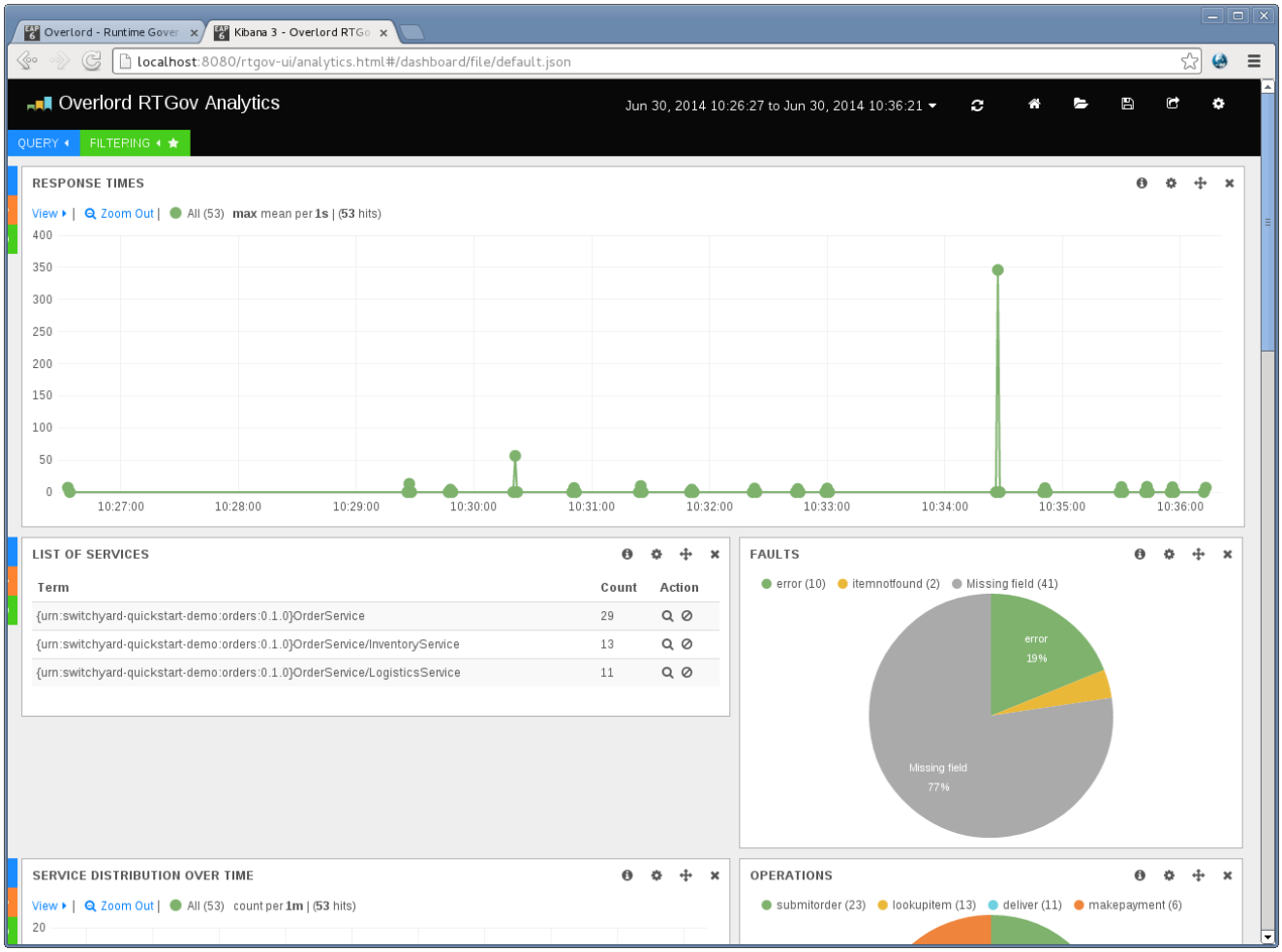
Overlord - Runtime Gover
Kibana 3 - Overlord RTGo
localhost:8080/rtgov-ui/analytics.html#/dashboard/file/default.json
Overlord RTGov Analytics
Jun 30, 2014 10:26:27 to Jun 30, 2014 10:36:21
QUERY
FILTERING
RESPONSE TIMES
View | Zoom Out | All (53) max mean per 1s | (53 hits)
400
350
300
250
200
150
100
50
0
10:27:00
10:28:00
10:29:00
10:30:00
10:31:00
10:32:00
10:33:00
10:34:00
10:35:00
10:36:00
LIST OF SERVICES
Term
Count
Action
{urn:switchyard-quickstart-demo:orders:0.1.0}OrderService
29
{urn:switchyard-quickstart-demo:orders:0.1.0}OrderService/InventoryService
13
{urn:switchyard-quickstart-demo:orders:0.1.0}OrderService/LogisticsService
11
FAULTS
error (10)
itemnotfound (2)
Missing field (41)
error
19%
Missing field
77%
SERVICE DISTRIBUTION OVER TIME
View | Zoom Out | All (53) count per 1m | (53 hits)
20
OPERATIONS
submitorder (23)
lookupitem (13)
deliver (11)
makepayment (6)

Figure 3.17.

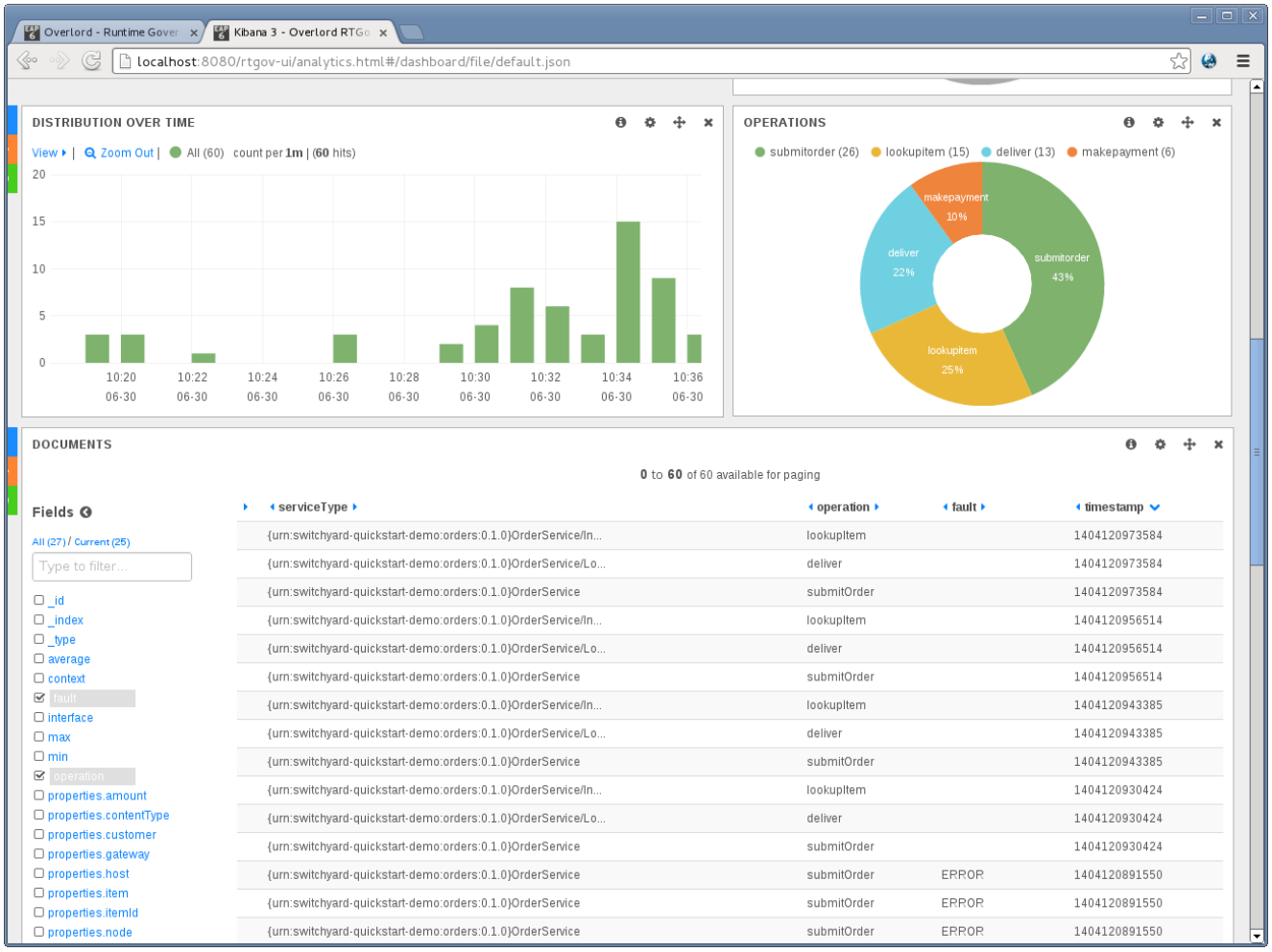

Figure 3.18.

3.4.1.1. Response Times

This graph displays response time information that matches any defined filter. Initially the only filter that is applied is a default time frame showing information over the last 24 hours (see following section on *Changing the Time Frame and Refresh Cycle*).

All response time information will be shown in the same (green) colour. This enables a general indication of performance to be obtained, but to identify specific issues it will be necessary to isolate response times of interest. This can be achieved using "Filtering by query" to only show response times within a particular range.

It is also possible to colour code response time information associated with particular subsets of the information (e.g. for particular service types, or customers, etc). See *Segmenting information by query* section for more information.

3.4.1.2. List of Services

This table shows the list of services. Each service is listed with the number of invocations (count) and actions that can be used to focus or exclude the particular service from the information being viewed.

Note

The service invocation count is based on the information available after all filters have been applied. This means it is possible to identify how many invocations of a particular service have been performed by setting the timeframe (see *Changing the Time Frame and Refresh Cycle*) or filters on other properties (e.g. customer, host, etc).

By default, the services listed in the table are related to *public* services. However if a service is marked as **internal**, then they will be excluded using the following filter:

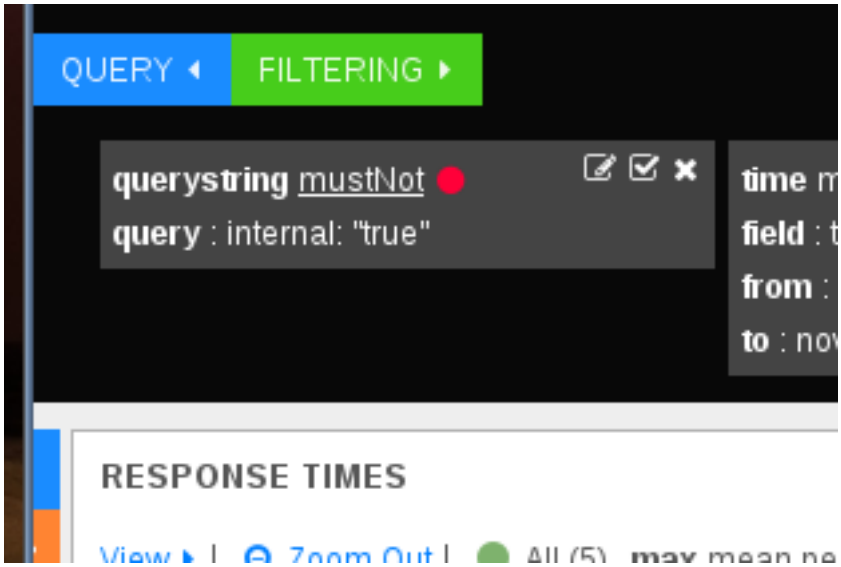

Figure 3.19.

To include the internal services in the service table, simply disable this filter by unchecking the filter.

3.4.1.3. Faults

This pie chart shows the distribution of faults that occur within the filtered response time information being viewed.

The segment labels mean:

- "Missing field" refers to response time information that had no associated fault

- "Error" is a general segment to identify response times associated with unnamed exceptions
- All other values are domain specific fault types (e.g. itemnotfound in this case).

Selecting a region from the pie chart will further focus the dashboard on response time information associated with that catagory of fault. To cancel the filter, select the "FILTERING" green tab at the top of the page, and either disable or remove the entry matching the fault filter.

3.4.1.4. Distribution over time

This bar charts shows the distribution of the response time information over time, grouped by a specified time interval (initially 1 minute).

When subsets of information are defined, based on *pinned queries*, it is possible to get more interesting results based on colour coded regions. For example, if separate queries are used to represent response times associated with different service types, then the bars will be colour coded to show how much activity occurred on each of the service types.

3.4.1.5. Operations

This pie chart is used to decompose the activities based on the operations that were performed, subject to any other filters that may have been applied.

It is also possible to create an additional filter on the currently viewed information, based on a particular operation, by selecting the operation of interest's segment within the pie chart. To cancel the filter, simply select the "FILTERING" green tab at the top of the page, and either disable or remove the entry matching the operation filter.

3.4.1.6. Documents

This section provides a list of the most recent response time information. The columns provide a small selection of fields from the response time events, with a list of the available fields as checkboxes down the left hand side. This enables the user to select additional fields of interest.

When a row is selected, it will expand to show the complete set of fields from the response time event, with some *action* icons next to each value. If the user selects the magnifying glass, then the dashboard will be additional focused on response time information associated with that field value, and similarly selecting the *no entry sign* will exclude information with that field value.

As mentioned previously, cancelling a particular filter can be achieved by selecting the "FILTERING" green tab at the top of the page, and either disable or remove the entry matching the field filter.

3.4.2. Changing the Time Frame and Refresh Cycle

The Kibana dashboard provides a mechanism for users to define the timeframe of interest, and the refresh interval.

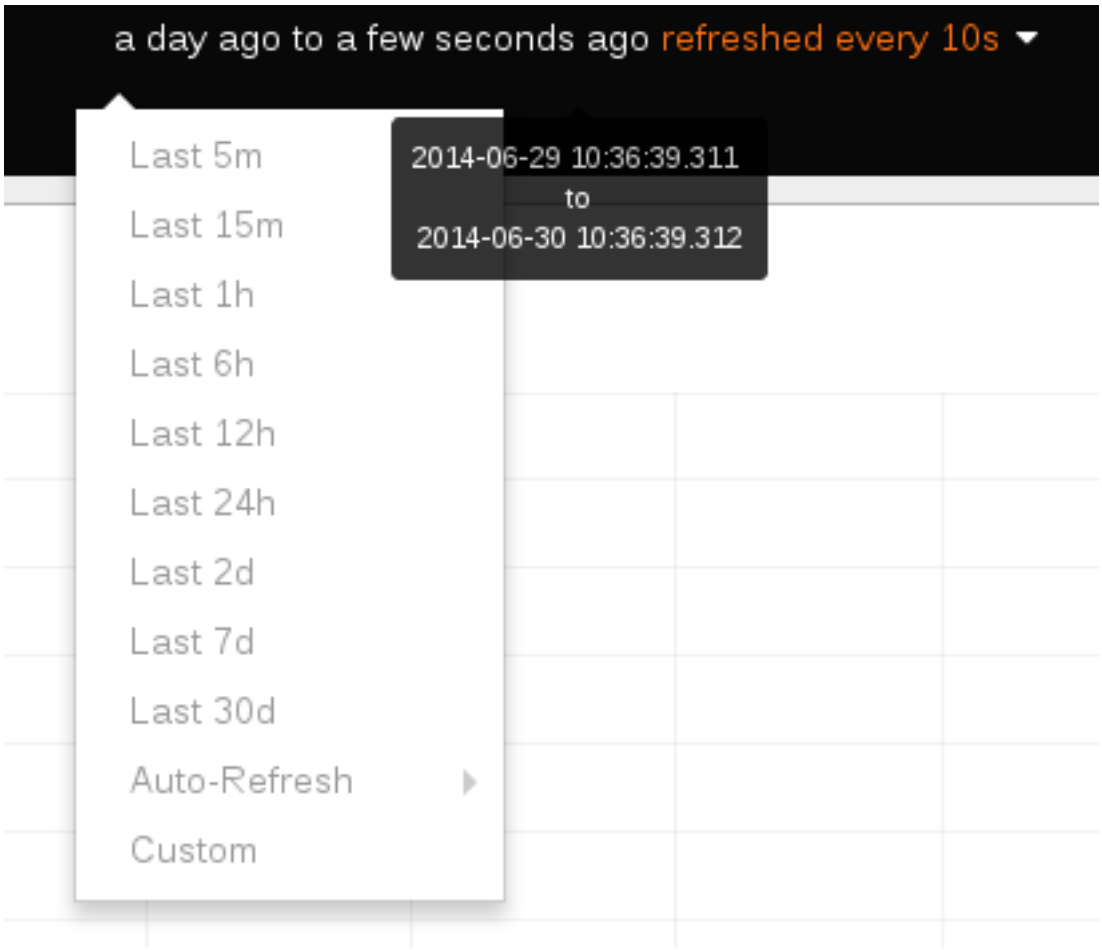

Figure 3.20.

The drop down menu at the top of the dashboard enables the user to select from a default set of time ranges in the past to the current time. If one of the default time ranges is not suitable, then a custom value can be selected.

Similarly, the refresh cycle can be selected from the values in the *Auto-Refresh* sub-menu, or alternatively disabled by selecting *Off*.

However it is also possible to interactively select a region from the response time graph (at the top of the page), to focus the attention of the dashboard on that time period. This creates a time based filter, which can be cancelled by selecting the "FILTERING" green tab at the top of the page, and either disable or remove the entry matching the time filter.

3.4.3. Filtering by selection

The Kibana dashboard enables a user to filter the information being viewed by:

- pressing the *magnifying glass* symbol associated with some information of interest (see action in the image below)

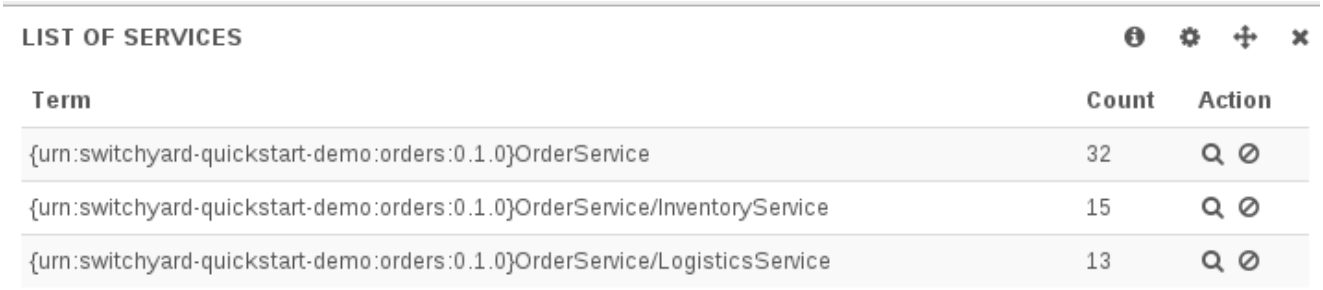

Figure 3.21.

- pressing the *no entry sign* symbol associated with the information to be excluded (see action in the image above)
- selecting the information of interest from a pie chart (e.g. selecting a fault, as shown in the image below)

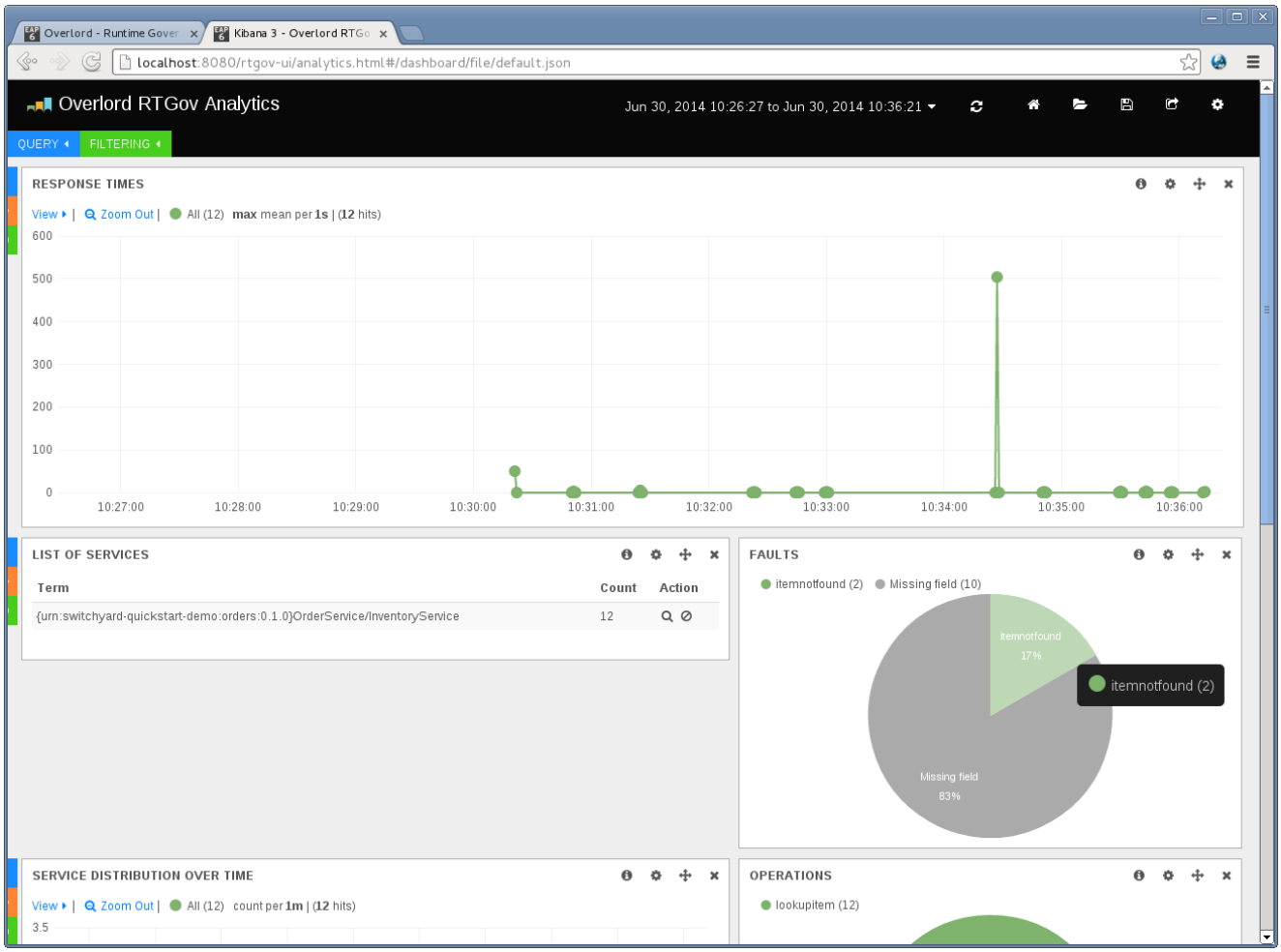

Figure 3.22.

As well as being able to focus/exclude information based on the other graphs, the *Documents* table provides even more fine grained control over what is displayed. In the following image it shows how the `fault` value of *itemnotfound* could be used as a filter, instead of selecting it from the pie chart. However, more importantly adhoc fields such as *customer* or *productName* could be equally used as the subject of the filter, if that information is recorded with the activity events (and therefore the response time data).

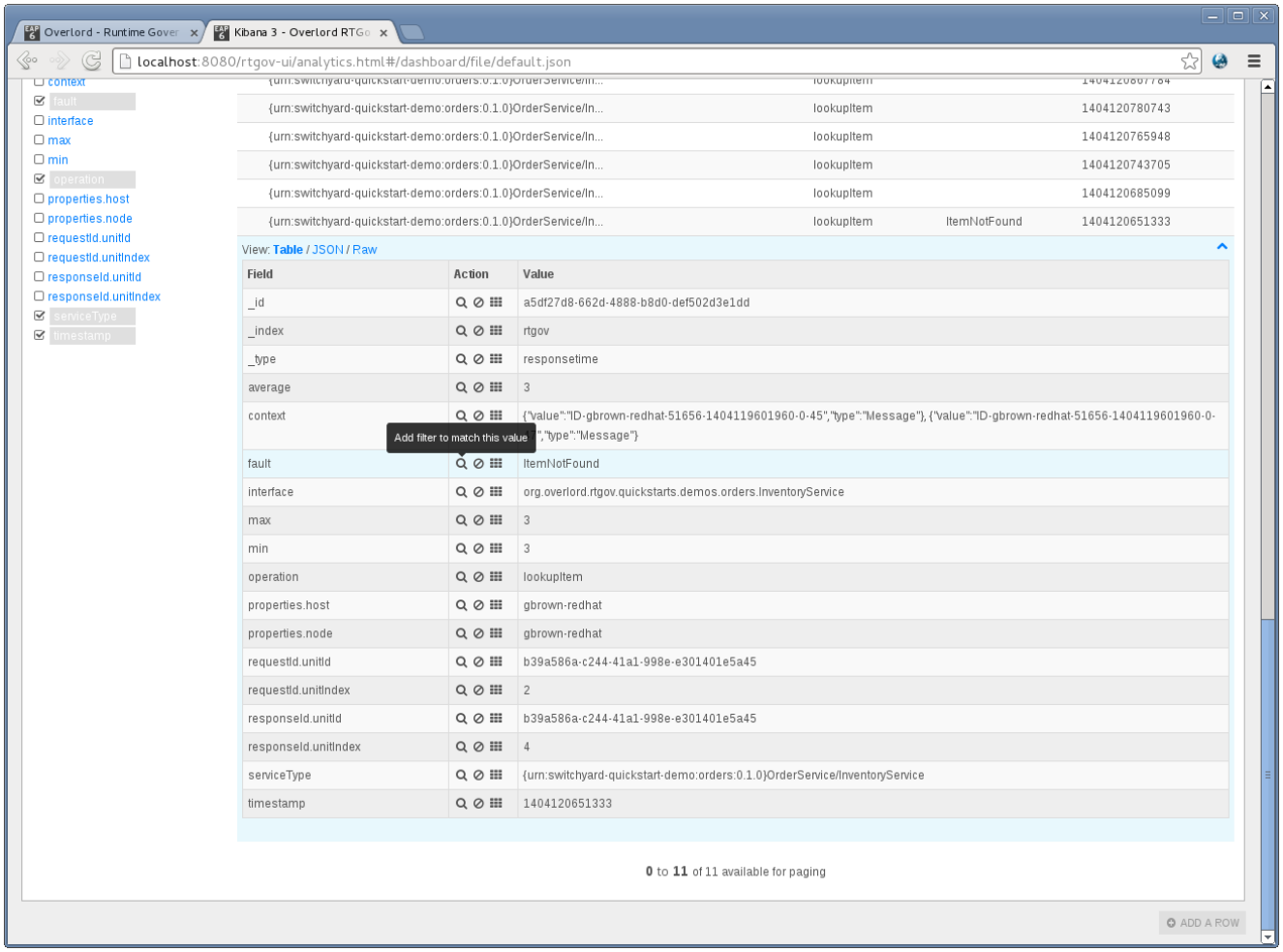

Figure 3.23.

As each filter is added, to progressively refine the results being viewed, their details are listed in the "FILTERING" section at the top of the dashboard, as shown in the following image:

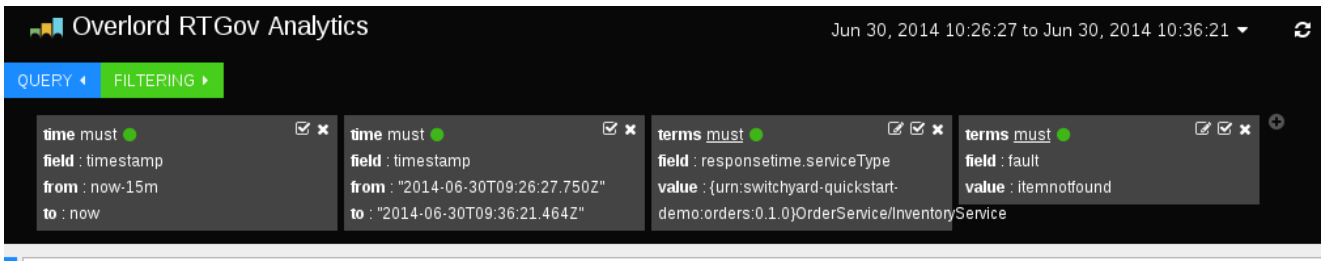

Figure 3.24.

The first box identifies the initial time range used to display the data, which has been refined by the next box based on interactively selecting a region on the response time graph. The third box applies a filter to only show information related to the *InventoryService* service type, and finally the fourth box narrows the information further to show the subset of response time information associated with the *itemnotfound* fault.

Any of the these filter criteria can individually be disabled (using the *tick* symbol) or cancelled (using the *cross* symbol).

3.4.4. Segmenting information by query

Although filtering provides a useful way to narrow in on information of interest to view that data in the available graphs. It is sometimes more interesting to be able to compare different sets of results.

In the default dashboard all response time information is treated in the same way, and therefore not differentiated. If we want to segment the information based on various groupings, then we need to create what are called *pinned queries*. At the top of the dashboard, you will need to expand the blue "QUERY" region to find a data entry area. This can be used to enter adhoc queries to filter the results displayed in the dashboard (see following section).

However for the purpose of comparing different sets of data, we leave the default entry blank and instead create one or more additional query fields, but pressing the *plus* symbol present in the last entry field.

When an entry field has been created, enter an appropriate query. For example,

- `serviceType: "{urn:switchyard-quickstart-demo-orders:0.1.0}OrderService/InventoryService"`

This query will identify response times associated with the *InventoryService* service type.

- `properties.customer: "Fred"`

If the customer name has been associated with the reported activity events, then this query will identify the response time information associated with a particular customer.

As shown in the following image, the colour coded segmented queries are reflected in the response time graph:

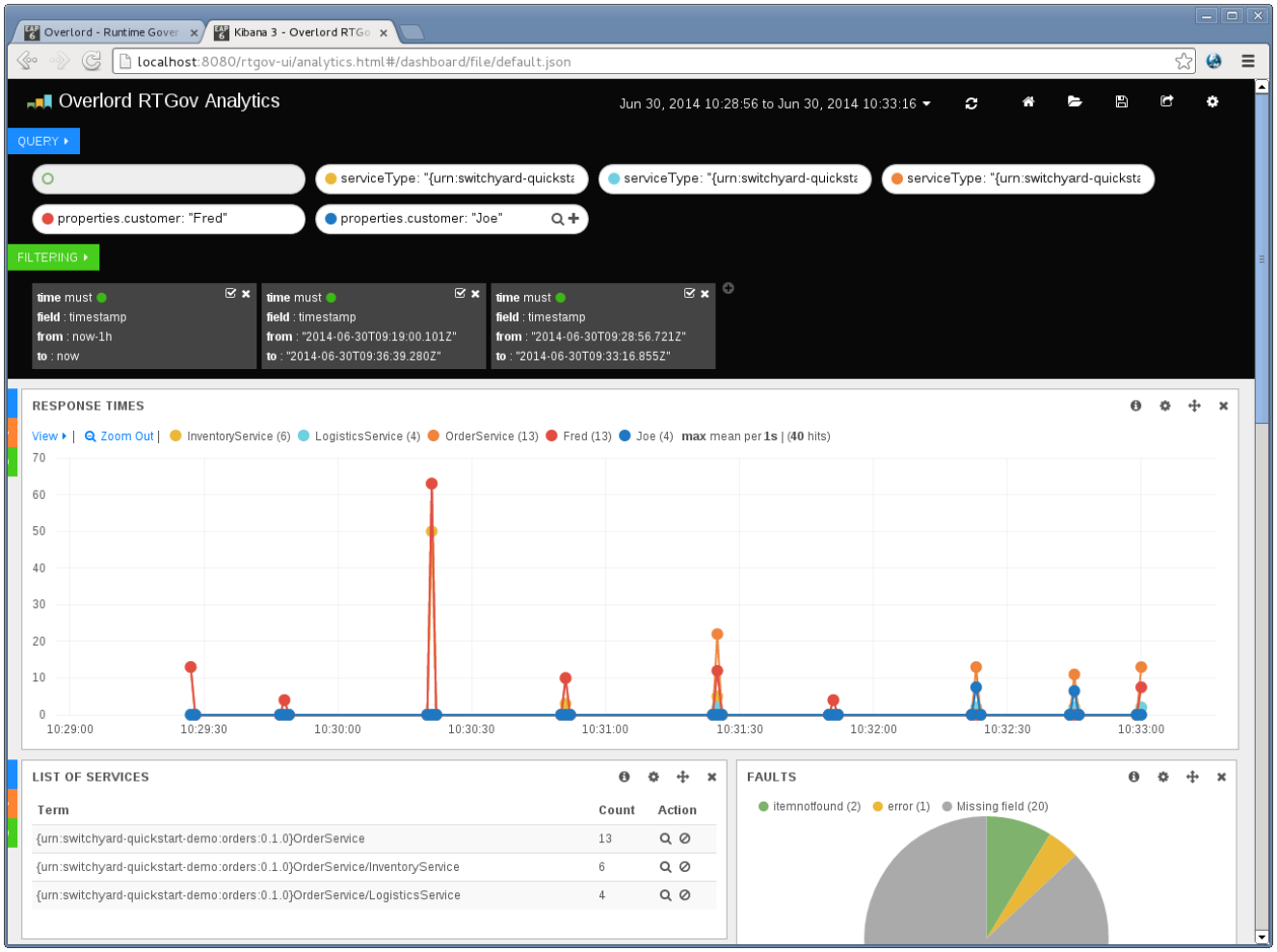

Figure 3.25.

as well as the *Distribution over time* chart:

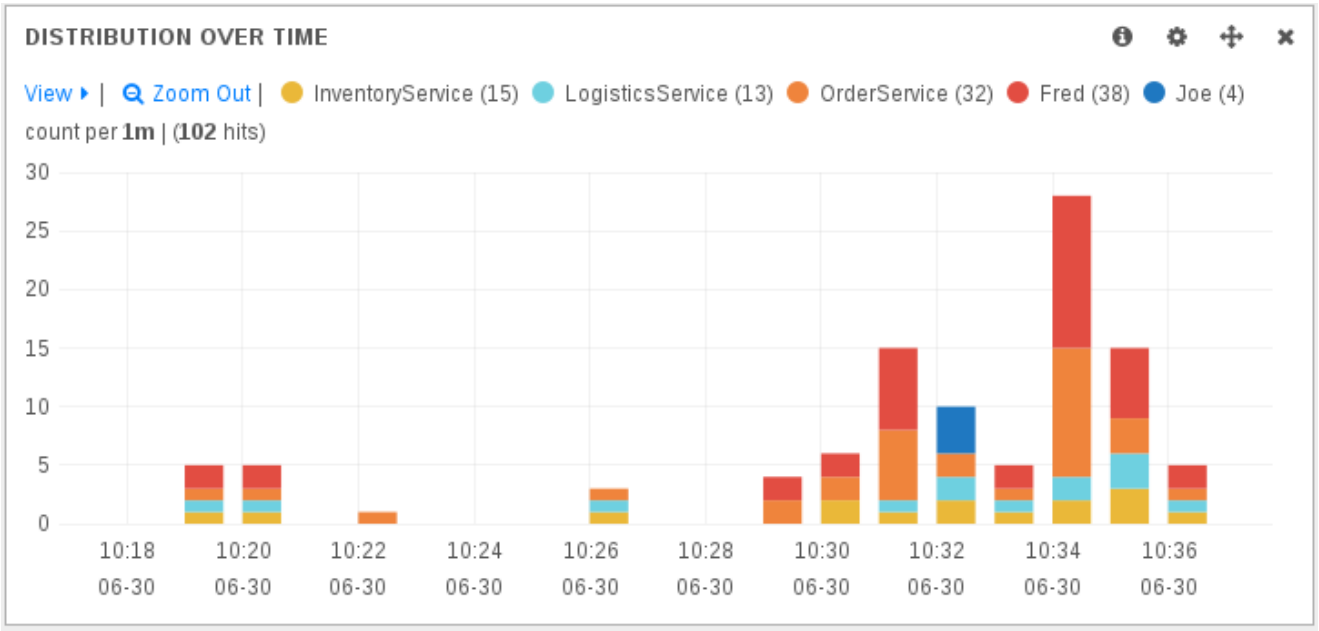

Figure 3.26.

To change the label associated with a query, select the query coloured dot and enter the label in the field, followed by pressing the close button:

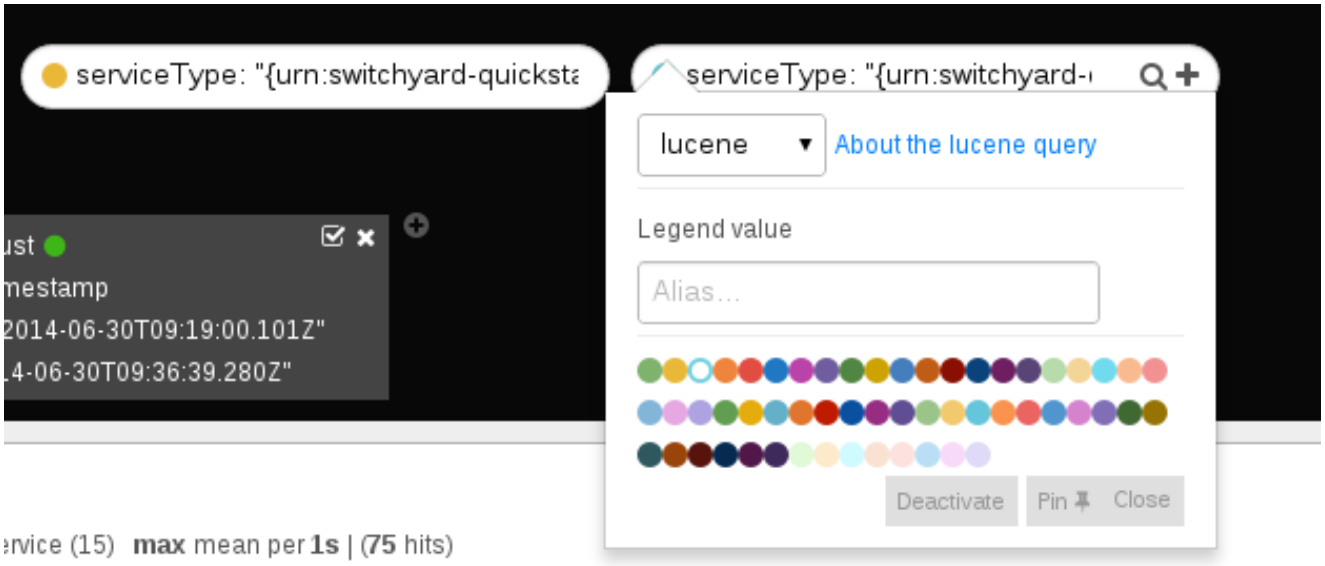

Figure 3.27.

It is also possible to temporarily disable a particular query, or change its colour, using this popup dialog.

3.4.5. Adhoc queries

Some times we need to focus the information on a particular property value or range. For example, if wanting to identify the services involved in increased response times, to locate potential

performance issues, then enter the query "max:>100" to show all response times that are greater than 100 milliseconds:

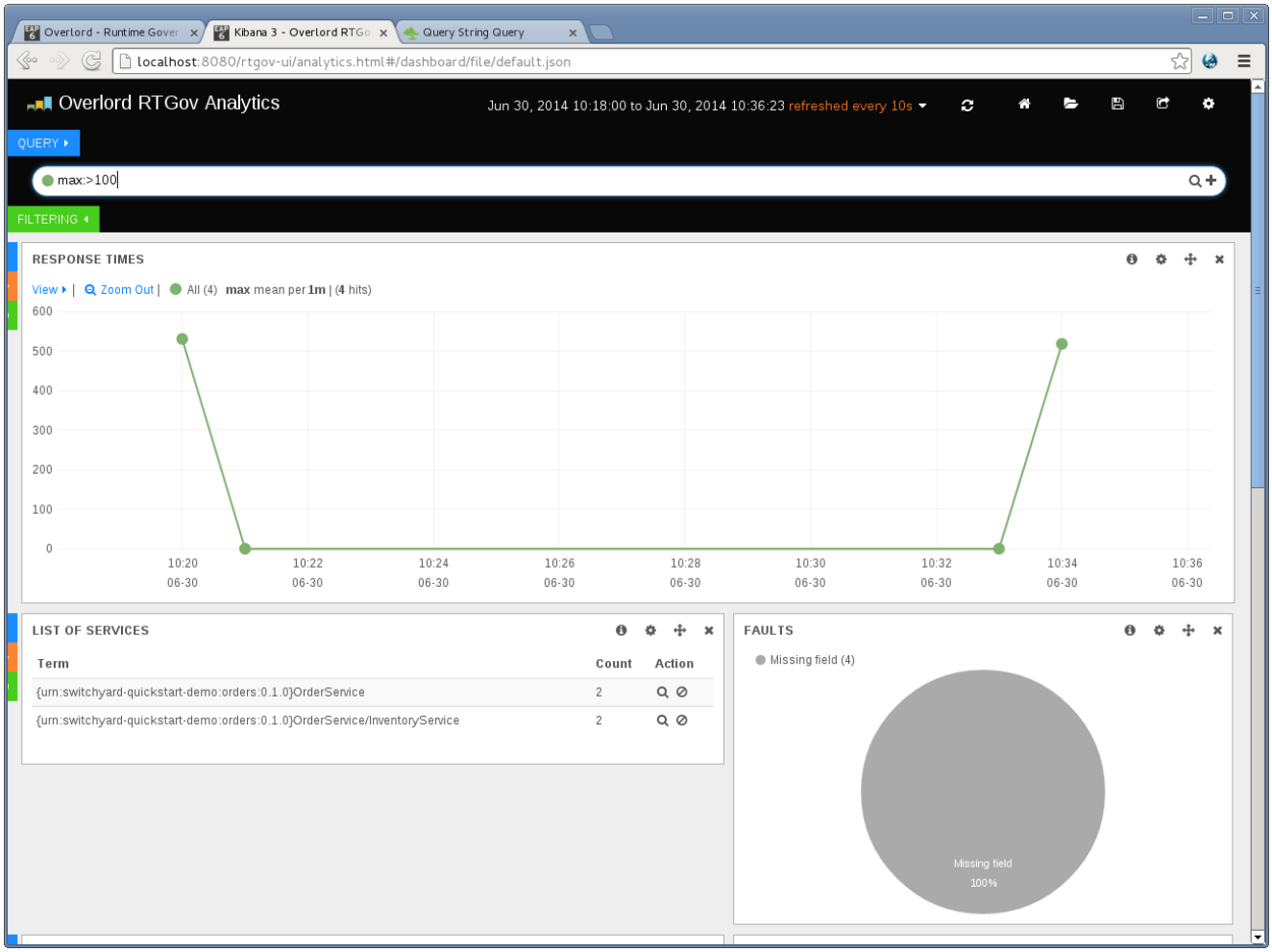

Figure 3.28.

Notice that the *List of Services* table now only includes the list of services that are related to those higher response times. The same applies to the *Operations* pie chart lower in the page. This can be used to pin point the services and operations that are causing the performance problems - and also by examining the *Documents* it is possible to identify other useful information, such as which customer was affected (if that information has been recorded with the activity events).

3.4.6. Customizing and sharing the Dashboard

It is possible to customise the dashboard, adding/removing rows and widgets within rows, configuring the graphs/charts/tables, etc.

Once a custom dashboard has been defined, then it can be saved using the *disc* symbol at the top right of the dashboard:

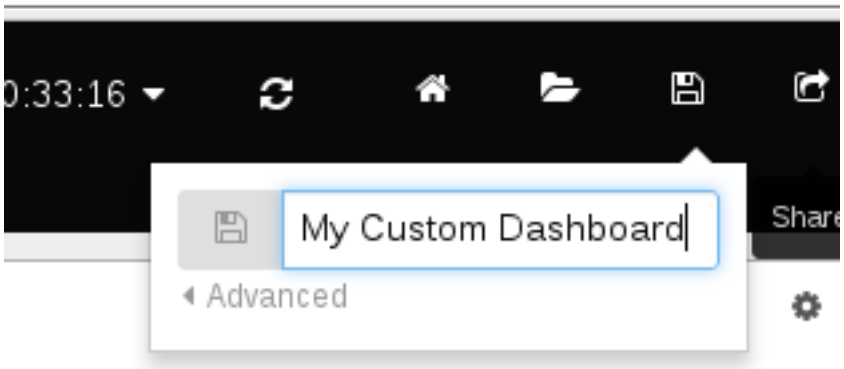

Figure 3.29.

When relaunching the RTGov UI, it is the possible to load a custom dashboard using:

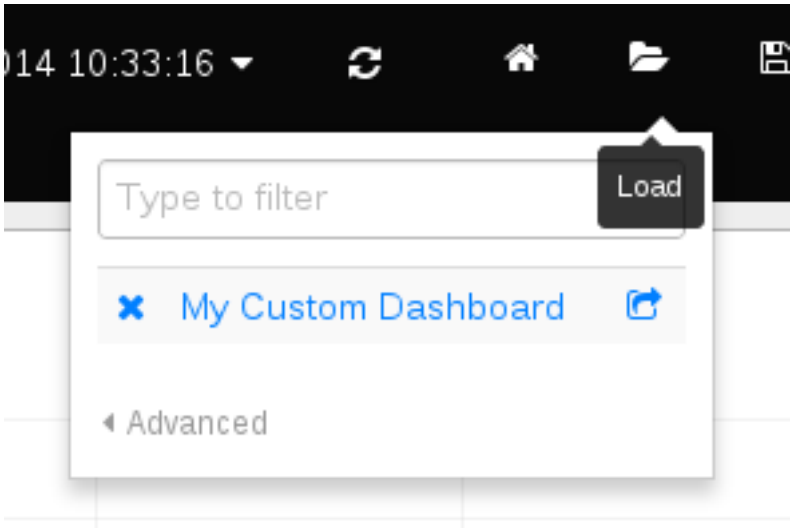

Figure 3.30.

It is also possible to export a custom dasbboard to a file, enabling it to be distributed to other interested users, who can then import it into their user account. Select the *save* icon, select *Advanced* and then *Export Schema*.

Chapter 4. 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.

4.1. Integrated Activity Collector

This section will discuss how an integrated activity collector can be used to automatically collect, pre-process and optionally validate activity events before finally reporting them to the server.

4.1.1. Supported Environments

This section discusses the environments that currently support integrated activity collectors.

4.1.1.1. SwitchYard

To collect activity events from a SwitchYard environment, simply install either the full server (if the execution and governance server are running co-located) or the client installation profile (if reporting events to another server).

4.1.1.2. OSGi Application

To collect activity events from an OSGi application, a proxy can be used to intercept inbound and outbound invocations on a service, and report the activity to an embedded activity collector within the OSGi container. These proxies can be wired into an OSGi application using blueprint, e.g.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

   <service
 interface="org.overlord.rtgov.quickstarts.demos.ordermgmt.orderservice.OrderService"
         ref="orderServiceProxy" />

   <bean id="orderServiceProxy"
 class="org.overlord.rtgov.client.ActivityProxyHelper"
```

```
       factory-method="createServiceProxy" >
     <argument
 value="org.overlord.rtgov.quickstarts.demos.ordermgmt.orderservice.OrderService" /
>
     <argument ref="orderServiceBean" />
    </bean>

    <bean id="orderServiceBean"
 class="org.overlord.rtgov.quickstarts.demos.ordermgmt.orderservice.OrderServiceBean"
 >
        <property name="inventoryService" ref="inventoryServiceProxy"/>
        <property name="logisticsService" ref="logisticsServiceProxy"/>
    </bean>

    <bean id="inventoryServiceProxy"
 class="org.overlord.rtgov.client.ActivityProxyHelper"
       factory-method="createClientProxy" >
     <argument
 value="org.overlord.rtgov.quickstarts.demos.ordermgmt.inventoryservice.InventoryService" /
>
     <argument ref="orderServiceBean" />
     <argument ref="inventoryServiceBean" />
    </bean>

    <bean id="logisticsServiceProxy"
 class="org.overlord.rtgov.client.ActivityProxyHelper"
       factory-method="createClientProxy" >
     <argument
 value="org.overlord.rtgov.quickstarts.demos.ordermgmt.logisticsservice.LogisticsService" /
>
     <argument ref="orderServiceBean" />
     <argument ref="logisticsServiceBean" />
    </bean>

    <reference id="inventoryServiceBean"

 interface="org.overlord.rtgov.quickstarts.demos.ordermgmt.inventoryservice.InventoryService
    </reference>

    <reference id="logisticsServiceBean"

 interface="org.overlord.rtgov.quickstarts.demos.ordermgmt.logisticsservice.LogisticsService
    </reference>

</blueprint>
```

The service interface is associated with a bean representing the service proxy, created using the **createServiceProxy** static factory method on the class *org.overlord.rtgov.client.ActivityProxyHelper*.

Similarly, the outbound relationships from the service to other OSGi components are established via a *client* proxy, using the **createClientProxy** static factory method on the *org.overlord.rtgov.client.ActivityProxyHelper* class.

4.1.2. 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 pre-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

Extracting the property information is important for various reasons:

- it enables the business policies to remain independent of the specific information format used, and thus more efficiently access the key details (i.e. as properties)
- it is important to control what information is distributed within the actvity events, for both size (i.e. performance) and security/privacy 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.

4.1.2.1. 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",
     "expression":"order/orderId"
    }
   }],
```

```
     "properties":[{
      "name":"customer",
      "evaluator":{
       "type":"xpath",
       "expression":"order/customer"
      }
     },{
      "name":"item",
      "evaluator":{
       "type":"xpath",
       "expression":"order/itemId"
      }
     }]
    },
    "{urn:switchyard-quickstart-demo:orders:1.0}submitOrderResponse":{
     "contexts":[{
      "type":"Conversation",
      "evaluator":{
       "type":"xpath",
       "expression":"orderAck/orderId"
      }
     }],
     "properties":[{
      "name":"customer",
      "evaluator":{
       "type":"xpath",
       "expression":"orderAck/customer"
      }
     },{
      "name":"total",
      "evaluator":{
       "type":"xpath",
       "expression":"orderAck/total"
      }
     }]
    },
    "java:org.switchyard.quickstarts.demos.orders.Order":{
     "contexts":[{
      "type":"Conversation",
      "evaluator":{
       "type":"mvel",
       "expression":"orderId"
      }
     }],
     "properties":[{
      "name":"customer",
      "evaluator":{
       "type":"mvel",
       "expression":"customer"
```

```
     }
    },{
     "name":"itemId",
     "evaluator":{
      "type":"mvel",
      "expression":"itemId"
     }
    }]
   },
   "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",
      "expression":"payment/customer"
     }
    },{
     "name":"amount",
     "evaluator":{
      "type":"xpath",
      "expression":"payment/amount"
     }
    }]
   },
   "{urn:switchyard-quickstart-demo:orders:1.0}makePaymentResponse":{
    "properties":[{
     "name":"customer",
```

```
      "evaluator":{
       "type":"xpath",
       "expression":"receipt/customer"
      }
     },{
      "name":"amount",
      "evaluator":{
       "type":"xpath",
       "expression":"receipt/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:

Field	Description
	Numeric - i.e. simply define the version as a number Dot Format - i.e. 1.5.1.Final Any alpha, numeric and symbols.
typeProcessors	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.
properties	The list of property evaluators.
script	An optional script evaluator that is used to do any other processing that may be required, such as setting additional properties in the activity event that are not necessarily derived from message content information.
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, Message or Link. These types are explained below.
timeframe	The number of milliseconds associated with a *Link* context type. If not specified, then the context is assumed to represent the destination of the link, so the source of the link must define the timeframe.

Field	Description
header	The optional header name. If not defined, then the expression will be applied to the information content to obtain the context value.
evaluator	The expression evaluator used to derived the context value. See further down for details.

The context types represent different ways in which the activity events can be related to each other or to a logical grouping (e.g. business transaction). Not all activity events need to be associated directly with a global business transaction id. They can be indirectly associated based on transitive correlation - e.g. activity 1 is associated with the global business transaction id, activity 2 is associated with activity 1 by a message context type, and activity 3 is associated with activity 2 based on an endpoint correlation id. All three activity events will be collectively correlated to the business transaction id.

An explanation of the different context types is,

Context Type	Explanation
Conversation	A conversation identifier can be used to correlate activity events to a business transaction associated with a globally unique identifer (e.g. an order id).
Endpoint	A globally unique identifier associated with one endpoint in a business transaction. For example, a process instance id associated with the business process executing within a service playing a particular role in the business transaction.
Message	The globally unique identify of a message being sent from one party to another.
Link	A temporal link between a source and destination activity. The temporal nature of the association is intended to enable non-globally unique details to be used to correlate activities, where the id is considered unique within the defined timeframe.

Property Evaluator

The fields associated with the Property Evaluator component are:

Field	Description
name	The property name being initialized.

Field	Description
header	The optional header name. If not defined, then the expression will be applied to the information content to obtain the property value.
evaluator	The expression evaluator used to derive the property value. See further down for details.

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**.
expression	The expression to evaluate.
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

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 can take one optional additional boolean field *includeHeaders* (with default value false). This transformer 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. If the optional *includeHeaders* field is specified as *true*, then any header values that accompany the message that are represented as either String or DOM, will be serialized in an internal property, which can then be used by tooling (e.g. the resubmission capability in the RTGov UI).

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":{
  ....
  "transformer":{
   "type":"serialize"
  }
 },
```

4.1.2.2. 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>
 .....
```

OSGi Container

The Information Processors are deployed within the OSGi container as a JAR file with the following structure:

```
jarfile
|
|-META-INF
|    |- MANIFEST.MF
|
|-ip.json
|-ip-loader-osgi.jar
|-<custom classes/resources>
|-<additional libraries>
```

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

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

If custom classes are defined, then any associated classes, resources and additional libraries are located in the top level 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>${rtgov.version}</version>
   <scope>provided</scope>
  </dependency>
  <dependency>
   <groupId>org.overlord.rtgov.activity-management</groupId>
   <artifactId>ip-loader-osgi</artifactId>
   <version>${rtgov.version}</version>
  </dependency>
  ....
 </dependencies>

 <build>
  <finalName>....</finalName>
  <resources>
   <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
   </resource>
  </resources>
  <plugins>
   <plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
     <extensions>true</extensions>
    <configuration>
     <instructions>
      <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
      <Bundle-Version>${project.version}</Bundle-Version>
```

```
      <Bundle-
Activator>org.overlord.rtgov.activity.processor.loader.osgi.IPActivator</
Bundle-Activator>
      <Import-Package>
       !javax.inject.*,!javax.enterprise.*,!javax.persistence.*,
                          ....,
       *
      </Import-Package>
      <Embed-Dependency>*;scope=compile|runtime</Embed-Dependency>
     </instructions>
    </configuration>
   </plugin>
  </plugins>
 </build>
</project>
```

4.1.3. Activity Validation

The Activity Validator 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.

In some execution environments these validators can be implicitly called as part of collecting the activity events. However in some environments these validators need to be explicitly invoked, as they impact the execution behaviour. The SwitchYard environment is an example of this later environment, where an *interceptor* needs to be explicitly included within the SwitchYard application, which is responsible for invoking the validation capability and reacting to any issues it detects. To see how to configure such an interceptor, please see the synchronous policy sample.

4.1.3.1. Defining the Activity Validators

The Activity Validator 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 Validators. 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.common.infinispan.service.InfinispanCacheManager"
      }
    }
  }
}]
```

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

Field	Description
name	The name of the Activity Validator.
version	The version of the Activity Validator. If multiple versions of the same named Activity Validator 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 Validator 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.

4.1.3.2. Registering the Activity Validators

JEE Container

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

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

The `av.json` file contains the JSON representation of the Activity Validator configuration.

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

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>av-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>
 .....
```

OSGi Container

The Activity Validators are deployed within the OSGi container as a JAR file with the following structure:

```
jarfile
|
|-META-INF
|    |- MANIFEST.MF
|
|-av.json
|-av-loader-osgi.jar
|-<custom classes/resources>
|-<additional libraries>
```

The `av.json` file contains the JSON representation of the Activity Validator configuration.

The `av-loader-osgi.jar` acts as a bootstrapper to load and register the Activity Validators.

If custom classes are defined, then any associated classes, resources and additional libraries can be located in the top level 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>${rtgov.version}</version>
   <scope>provided</scope>
  </dependency>
  <dependency>
   <groupId>org.overlord.rtgov.activity-management</groupId>
   <artifactId>av-loader-osgi</artifactId>
   <version>${rtgov.version}</version>
  </dependency>
  ....
 </dependencies>

 <build>
  <finalName>....</finalName>
  <resources>
   <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
   </resource>
  </resources>
  <plugins>
   <plugin>
    <groupId>org.apache.felix</groupId>
```

```
    <artifactId>maven-bundle-plugin</artifactId>
     <extensions>true</extensions>
    <configuration>
     <instructions>
      <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
      <Bundle-Version>${project.version}</Bundle-Version>
      <Bundle-
Activator>org.overlord.rtgov.activity.validator.loader.osgi.AVActivator</
Bundle-Activator>
      <Import-Package>
       !javax.inject.*,!javax.enterprise.*,!javax.persistence.*,
                            .....;
       *
      </Import-Package>
      <Embed-Dependency>*;scope=compile|runtime</Embed-Dependency>
     </instructions>
    </configuration>
   </plugin>
  </plugins>
 </build>

</project>
```

4.2. 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.

4.2.1. Reporting Activity Information

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

The service uses basic authentication, with the default username `admin` and password `overlord`.

The request contains the list of ActivityUnit objects encoded in JSON. (See `org.overlord.rtgov.activity.model.ActivityUnit` class within the API documentation, as the root component of this configuration). 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",
        "principal":"Fred",
        "node":"Saturn1",
        "thread":"Thread-1"
    }
},{
    .....
}]
```

4.2.2. Querying Activity Events using an Expression

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

The service uses basic authentication, with the default username `admin` and password `overlord`.

The request contains the JSON encoding of the Query Specification (see API documentation for +org.overlord.rtgov.activity.server.QuerySpec+) which has the following properties:

Property	Description
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 only supported format currently is "jpql" (Java Persistence Query Language).

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. (See API documentation for `org.overlord.rtgov.activity.model.ActivityType`).

4.2.3. Retrieving an Activity Unit

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

The service uses basic authentication, with the default username `admin` and password `overlord`.

The <unitId> represents the identifier associated with the ActivityUnit that is being retrieved encoded in JSON. (See API documentation for `org.overlord.rtgov.activity.model.ActivityUnit`).

4.2.4. Retrieve Activity Events associated with a Context Value

GET request to URL: `<host>/overlord-rtgov/activity/events?type=<contextType>&value=<identifier>`

The service uses basic authentication, with the default username `admin` and password `overlord`.

The <contextType> represents the context type, e.g. Conversation, Endpoint, Message or Link. This is explained in the Information Processor section of this chapter, or see the API documentation for `org.overlord.rtgov.activity.model.Context.Type`.

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

Two additional optional query parameters can be provided, `start` being the start timestamp, and `end` for the end timestamp. These parameters can be used to scope the time period of the query.

The response is a list of ActivityType objects (see `org.overlord.rtgov.activity.model.ActivityType` in the API documentation) encoded in JSON.

Chapter 5. Analyzing Events

5.1. 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.

5.1.1. 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.Final",
  "subscriptions" : [ {
    "nodeName" : "SOAEvents",
    "subject" : "ActivityUnits"
  },
  {
    "nodeName" : "ServiceDefinitions",
    "subject" : "ActivityUnits"
  },
  {
    "nodeName" : "SituationsStore",
    "subject" : "Situations"
  } ],
  "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"
      } ]
    },{
      "name" : "SituationsStore",
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.jpa.JPAEventProcessor",
        "entityManager" : "overlord-rtgov-epn-non-jta"
      }
    }
  ]
}
```

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

```
{
  "name" : "AssessCreditPolicyEPN",
  "version" : "${project.version}",
  "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.common.infinispan.service.InfinispanCacheManager"
          }
        },
        "parameters" : {
         "creditLimit" : 150
        }
      }
    }
  ]
}
```

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.

5.1.1.1. 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.

5.1.1.2. 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

Field	Description
	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 the event process implementation to use. Other general fields that can be configured are, the map of services and the map of parameters 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).

5.1.2. Registering the Network

5.1.2.1. 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>
  <resources>
   <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
   </resource>
  </resources>
  <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>
 .....
```

5.1.2.2. OSGi Container

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

```
jarfile
|
|-META-INF
|    |- MANIFEST.MF
|
|-epn.json
|-epn-loader-osgi.jar
|-<custom classes/resources>
|-<additional libraries>
```

The `MANIFEST.MF` file is important, as it contains the OSGi metadata required for the container to understand the contents and imported packages.

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

The `epn-loader-osgi.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, resources and additional libraries can be located in the top level 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>${rtgov.version}</version>
   <scope>provided</scope>
  </dependency>
  <dependency>
   <groupId>org.overlord.rtgov.event-processor-network</groupId>
   <artifactId>epn-loader-osgi</artifactId>
   <version>${rtgov.version}</version>
  </dependency>
  ....
 </dependencies>

 <build>
  <finalName>....</finalName>
  <resources>
   <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
   </resource>
  </resources>
  <plugins>
   <plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
     <extensions>true</extensions>
    <configuration>
     <instructions>
      <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
      <Bundle-Version>${project.version}</Bundle-Version>
```

```
      <Bundle-Activator>org.overlord.rtgov.epn.loader.osgi.EPNActivator</
Bundle-Activator>
      <Import-Package>
       !javax.inject.*,!javax.enterprise.*,!javax.persistence.*,
                           ....,
       *
      </Import-Package>
      <Embed-Dependency>*;scope=compile|runtime</Embed-Dependency>
     </instructions>
    </configuration>
   </plugin>
  </plugins>
 </build>
</project>
```

5.1.3. 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.

5.2. Event Processors

As previously mentioned, all EventProcessor implementations can define the following information:

Field	Description
services	The optional map of names to services. The current service types are listed at the bottom of this section.
parameters	The optional map of names to parameters. These parameters can be used to customize the behaviour of an event processor.
asynchronous	This optional and **experimental** boolean flag enables an event processor to produce its results asynchronously. This has been added

Field	Description
	to support CEP, and currently means any results are processed as individual events which may be less efficient.

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

5.2.1. Drools Event Processor

The Drools Event Processor implementation (`org.overlord.rtgov.ep.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".
eventProcessingMode	This optional field identifies the event processing mode. Valid values are *cloud* (default) and *stream*. If *stream* is chosen, then you will also need to set the *asynchronous* property to *true*.
clockType	The optional clock type. Valid values are *realtime* (default) and *pseudo*.

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
```

This component 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).

The rule can also access parameters using the getParameter(name) method on the context. See the javadoc for the `org.overlord.rtgov.ep.EPContext` interface for more information.

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.

5.2.2. JPA Event Processor

A JPA based Event Processor implementation (`org.overlord.rtgov.ep.jpa.JPAEventProcessor`) enables events to be persisted. This implementation defines the following additional fields:

Field	Description
entityManager	The name of the entity manager to be used.

5.2.3. Mail Event Processor

A mail based Event Processor implementation (`org.overlord.rtgov.ep.mail.MailEventProcessor`) enables events to be transformed and sent as an email. This implementation defines the following additional fields:

Field	Description
from	The *from* email address.
to	The list of *to* email addresses.
subjectScript	The location of the MVEL script, which may be relative to the classpath, used to define the email subject.
contentScript	The location of the MVEL script, which may be relative to the classpath, used to define the email content.
contentType	The optional type of the email content. By default it will be "text/plain".
jndiName	The optional JNDI name locating the JavaMail session.

5.2.4. MVEL Event Processor

A MVEL based Event Processor implementation (`org.overlord.rtgov.ep.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, *getParameter* method for obtaining custom properties, and various logging methods.

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.

5.2.5. 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.

5.2.5.1. 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

Infinispan

Class name: `org.overlord.rtgov.common.infinispan.service.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 a default cache manager will be created.

5.3. Predicates

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

5.3.1. MVEL Predicate

A MVEL based Predicate implementation (`org.overlord.rtgov.ep.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 6. Accessing Derived Information

6.1. Configuring Active Collections

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.

6.1.1. 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",
    "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",
    "activeChangeListeners" : [ {
        "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
     "objectName" : "overlord.rtgov.services:name=Situations",
     "descriptionScript" : "SituationDescription.mvel",
     "insertTypeScript" : "SituationType.mvel"
} ],
"derived": [ {
 "name": "FilteredSituations",
 "predicate": {
  "type": "MVEL",
  "expression": "map = context.getMap(\"IgnoredSituationSubjects\"); if
(map == null) { return false; } return !map.containsKey(subject);"
 },
    "properties" : {
        "active" : false
    }
} ]
 },{
    "@class" :
"org.overlord.rtgov.active.collection.ActiveCollectionSource",
    "name" : "IgnoredSituationSubjects",
    "type" : "Map",
    "lazy" : true,
    "factory" : {
        "@class" :
"org.overlord.rtgov.active.collection.infinispan.InfinispanActiveCollectionFactory",
        "cache" : "IgnoredSituationSubjects"
    }
 },{
    "@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 multiple 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 Active Collection that will be created and associated with this source.
type	The type of active collection. The currently supported values (as defined in the `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.

Field	Description
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 aggregate 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 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.
derived	An optional list of definitions for derived collections that will be created with the top level active collection, and retained regardless of whether any users are currently accessing them. (Normally when a derived collection is created dynamically on demand, once it has served its purpose, it will be cleaned up). The definition will be explained below.
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.AbstractActiveChan`
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.

6.1.1.1. Scripts

Aggregation

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 aggregated result will be returned from the script.

Scheduled

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.

Maintenance

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.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.

6.1.1.2. Derived Active Collections

The *derived* element defines a list of derived active collection definitions that will be instantiated with the active collection.

The fields associated with this component are:

Field	Description
name	The derived active collection's name.
predicate	The predicate that will determine what subset of entries from the parent collection should be available within the derived collection.
properties	Properties that will be passed to the derived active collection.

The following properties can be defined:

Property	Description
active	This optional property indicates whether the derived collection should be actively maintained (i.e. active = true), which is the default, or whether the contents should be determined when a query is performed. The main reason for setting this property to false is due to the predicate being based on volatile information, and therefore the contents needs to be evaluated at the time it is requested.

6.1.1.3. 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.

6.1.1.4. 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 a default cache manager will be instantiated.

6.1.2. Registering the Source

6.1.2.1. 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>
 .....
```

6.1.2.2. OSGi Container

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

```
jarfile
|
|-META-INF
|    |- beans.xml
|
|-acs.json
|-acs-loader-osgi.jar
|-<custom classes/resources>
|-<additional libraries>
```

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

The `acs-loader-osgi.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, resources and additional libraries can be located in the top level 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>${rtgov.version}</version>
   <scope>provided</scope>
  </dependency>
  <dependency>
   <groupId>org.overlord.rtgov.active-queries</groupId>
   <artifactId>acs-loader-osgi</artifactId>
   <version>${rtgov.version}</version>
  </dependency>
  ....
 </dependencies>

 <build>
  <finalName>....</finalName>
  <resources>
   <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
   </resource>
  </resources>
  <plugins>
   <plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
     <extensions>true</extensions>
    <configuration>
```

```
     <instructions>
      <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
      <Bundle-Version>${project.version}</Bundle-Version>
      <Bundle-Activator>org.overlord.rtgov.acs.loader.osgi.ACSActivator</
Bundle-Activator>
      <Import-Package>
       !javax.inject.*,!javax.enterprise.*,!javax.persistence.*,
                            ....,
                            *
      </Import-Package>
      <Embed-Dependency>*;scope=compile|runtime</Embed-Dependency>
     </instructions>
    </configuration>
   </plugin>
  </plugins>
 </build>

</project>
```

6.2. 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.services:name=Situations",
     "descriptionScript" : "SituationDescription.mvel",
     "insertTypeScript" : "SituationType.mvel"
    } ],
    "derived": [ {
        "name": "FilteredSituations",
        "predicate": {
            "type": "MVEL",
            "expression": "map = context.getMap(\"IgnoredSituationSubjects
\"); if (map == null) { return false; } return !map.containsKey(subject);"
```

```
        },
        "properties" : {
            "active" : false
        }
    } ]
  }
]
```

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",
  "version" : "${project.version}",
  "subscriptions" : [ {
    "nodeName" : "SLAViolations",
    "subject" : "ServiceResponseTimes"
  } ],
  "nodes" : [
    {
      "name" : "SLAViolations",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ "Situations" ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.drools.DroolsEventProcessor",
        "ruleName" : "SLAViolation",
        "parameters" : {
         "levels" : [
          {
           "threshold" : 400,
           "severity" : "Critical"
          },
          {
           "threshold" : 320,
           "severity" : "High"
          },
          {
           "threshold" : 260,
           "severity" : "Medium"
```

```
          },
          {
           "threshold" : 200,
           "severity" : "Low"
          }
         ]
        }
      },
      "predicate" : null,
      "notifications" : [ {
        "type" : "Processed",
        "subject" : "SituationsProcessed"
      },{
        "type" : "Results",
        "subject" : "Situations"
      } ]
    }
  ]
}
```

6.3. 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.

Field	Description
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:

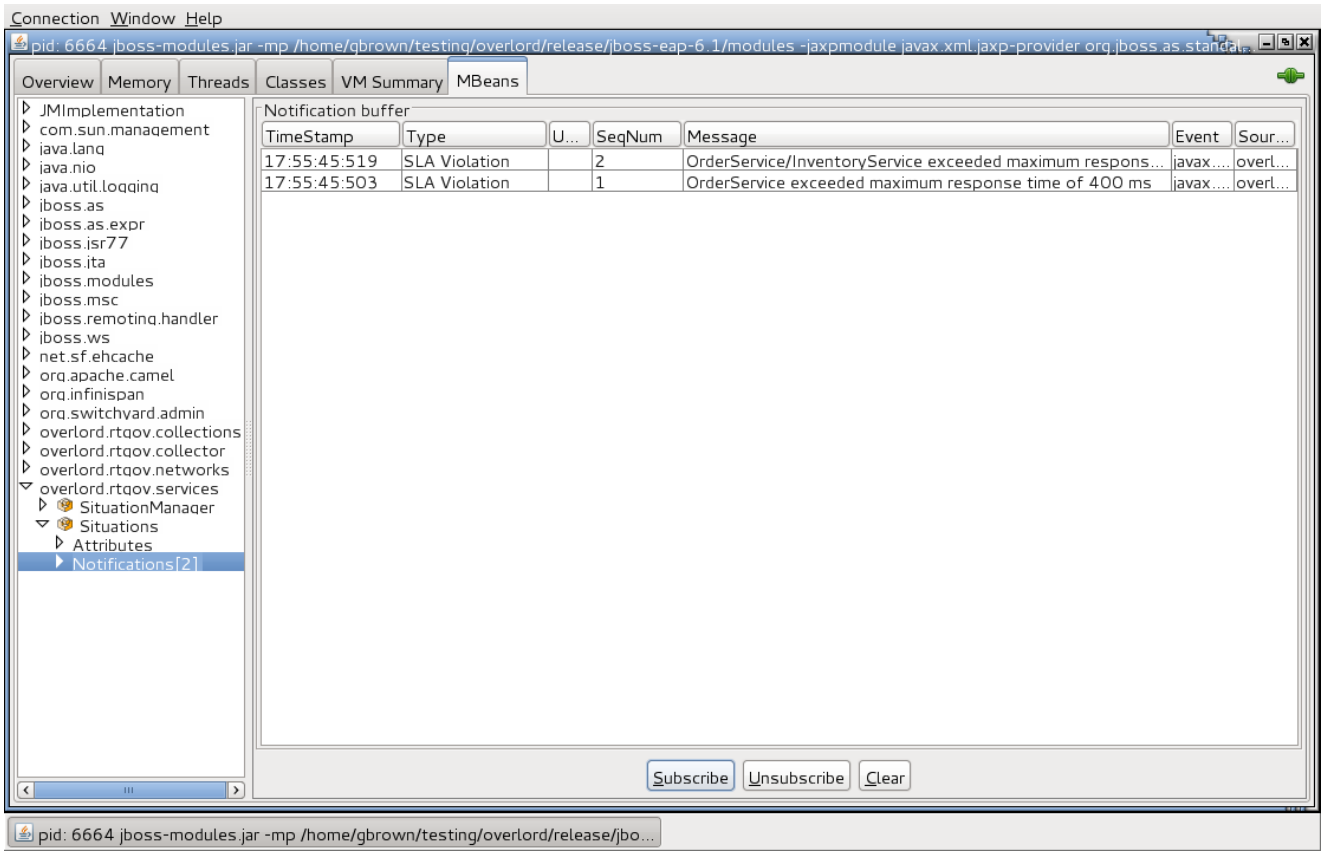

Figure 6.1.

6.4. Querying Active Collections via REST

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

This service used basic authentication, with a default username `admin` and password `overlord`.

The Query Specification (see `org.overlord.rtgov.active.collection.QuerySpec` in the API documentation) 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.
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.
properties	Map of key/value pairs, used when creating a derived collection. Currently the only relevant property is a boolean called *active*, defaults to true, which can be used to force queries on the derived collection to be evaluated when information requested, in situations where the predicate is based on volatile information.

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" : "ServiceResponseTimes",
    "maxItems" : 5000,
    "collection" : "OrderServiceSRT",
    "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 *OrderServiceSRT*, 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.

The list of objects returned by the query will be represented in JSON.

6.5. Pre-Defined Active Collections

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

6.5.1. 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. For more details please see the API documentation.

6.5.2. Situations

This active collection is a `list` of `org.overlord.rtgov.analytics.situation.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. For more information on this class, please see the API documentation.

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"
    } ],
    .......
  }
]
```

6.5.3. ServiceDefinitions

This active collection is a `map` of Service Type name to `org.overlord.rtgov.analytics.service.ServiceDefinition` objects. More details on this class can be found in the API documentation.

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*.

6.5.4. 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 7. Available Services

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

7.1. 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?type=<type>&value=<value>`

The service uses basic authentication, with a default username `admin` and password `overlord`.

This service has the following query parameters:

Parameter	Description
type	The type of the identify value, e.g. Conversation, Endpoint, Message or Link
value	The identifier value, e.g. if type is Conversation, then the value would be a globally unique identifier for the business transaction

The call trace is returned as a JSON representation of the call trace object model. The top level class is `org.overlord.rtgov.call.trace.model.CallTrace`, details can be found in the API documentation.

7.2. Report Server

Note

As of version 2.0, the report server is deprecated, indicating that it will be removed from the project in a future release. Therefore we suggest that you should not start using it, and if already using it, plan to migrate off this capability in the new future. The capability is being deprecated as more sophisticated analysis and reporting can be achieved through the Kibana/Elasticsearch tools, which are now integrated from RTGov 2.0.

The "Report Server" service is used to generate instances of a report whose definition has previously been deployed to the server. This section will explain how to configure and deploy a report definition, and then how to generate the report instances.

7.2.1. Creating and deploying a report definition

The first step is to specify a JSON representation of the `org.overlord.rtgov.reports.ReportDefinition` class (see API documentation for details).

```
[
  {
    "name" : "SLAReport",
    "generator" : {
      "@class" : "org.overlord.rtgov.reports.MVELReportGenerator",
       "scriptLocation" : "SLAReport.mvel"
    }
  }
]
```

The report definition only contains the `name` of the report, and the definition of the `generator`. In this case, the `org.overlord.rtgov.reports.MVELReportGenerator` implementation of the report generator has been used, which also includes a property to define the location of the report script (e.g. `SLAReport.mvel`). This MVEL SLA report script can be found in the `samples/sla/report` folder.

7.2.1.1. Registering the Report

JEE Container

The Report Definition is deployed within the JEE container as a WAR file with the following structure:

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

As described above, the `reports.json` file contains the JSON representation of the report definition configuration.

The `reports-loader-jee.jar` acts as a bootstrapper to load and register the Report Definition.

If custom report generators or scripts 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-analysis</groupId>
   <artifactId>reports-loader-jee</artifactId>
   <version>${project.version}</version>
  </dependency>
  <dependency>
   <groupId>org.overlord.rtgov.activity-analysis</groupId>
   <artifactId>reports</artifactId>
   <version>${project.version}</version>
   <scope>test</scope>
  </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>
 .....
```

7.2.2. Generating an instance of the report

The URL for the service's REST GET request is: `<host>/overlord-rtgov/report/generate?<parameters>`

The service uses basic authentication, with a default username `admin` and password `overlord`.

This service has the following query parameters:

Parameter	Description
report	The name of the report to be generated. This must match the previously deployed report definition name.
startDay/Month/Year	The optional start date for the report. If not defined, then the report will use all activities stored up until the end date.
endDay/Month/Year	The optional end date for the report. If not defined, then the report will use all activities up until the current date.
timezone	The optional timezone.
calendar	The optional business calendar name. A default called exists called *Default* which represents a working week of Monday to Friday, 9am to 5pm, excluding Christmas Day.

All other query parameters that may be provided will be specific to the report definition being generated.

The operation returns a JSON representation of the `org.overlord.rtgov.reports.model.Report` class. See the API documentation for further details of the object model.

7.2.3. Providing a custom Business Calendar

A custom Business Calendar can be defined as a JSON representation of the `org.overlord.rtgov.reports.mode.Calendar` class (see API documentation for details). This should be stored in a file whose location is referenced using a property called "calendar.<CalendarName>" in the `overlord-rtgov.properties` file.

7.3. 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>`

The service uses basic authentication, with a default username `admin` and password `overlord`.

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).

7.3.1. How to customize the severity levels

The severity levels used for the graph nodes and links can be customized by creating a MVEL script. A default script is provided within the `overlord-rtgov.war`, which can be used as a template. The script is called `SeverityAnalyzer.mvel` and is located within the `/WEB-INF/classes` folder of the `overlord-rtgov.war` archive.

An example of the contents of this script is:

```
Severity severity=Severity.Normal;

if (summary != null && latest != null && summary.getAverage() > 0) {
 double change=latest.getAverage()/summary.getAverage();

 if (change > 0) {

     if (change > 3.0) {
         severity = Severity.Critical;
```

```
      } else if (change > 2.2) {
          severity = Severity.Serious;
      } else if (change > 1.8) {
          severity = Severity.Error;
      } else if (change > 1.4) {
          severity = Severity.Warning;
      } else if (change > 1.2) {
          severity = Severity.Minor;
      }
  }
 }

 return (severity);
```

The script returns a value of type `org.overlord.rtgov.service.dependency.presentation.Severity`, which is automatically available as an imported class for use by the script.

The script takes four variables:

Variable	Description
summary	The summary metric to be evaluated.
history	The list of recent metrics, merged to produce the summary metric.
latest	The latest metric.
component	The service definition component associated with the metric. This variable is not used within the example script above.

If a customized script is created, then its location can be specified in the `MVELSeverityAnalyzer.scriptLocation` property in the `overlord-rtgov.properties` configuration file.

7.4. Situation Manager

The "Situation Manager" service is used to determine whether situations associated with a particular subject (i.e. service) should be displayed to users via the Situations gadget. The service supports two operations.

The service uses basic authentication, with a default username `admin` and password `overlord`.

7.4.1. Ignoring situations related to a subject

The `ignore` operation is used to indicate that situations for a particular subject (i.e. generally a service type) should not be presented to users via the REST service (and therefore the Situations gadget).

The URL for the `ignore` operation's POST request is: `<host>/overlord-rtgov/situation/manager/ignore`

This request supplies a JSON representation of the `org.overlord.rtgov.analytics.situation.IgnoreSubject` class. See the API documentation for more information.

The operation responds with a status message indicating whether the operation was successful.

Note

Currently wildcards are not supported for subjects.

7.4.2. Observing situations related to a subject

The `observe` operation is used to essentially reverse the actions performed by a previous `ignore` operation, to make situations for a particular subject (i.e. generally a service type) visible again to users via the REST service (and therefore the Situations gadget).

The URL for the `observe` operation's POST request is: `<host>/overlord-rtgov/situation/manager/observe`

This request supplies a JSON representation of the `org.overlord.rtgov.analytics.situation.IgnoreSubject` class. See the API documentation for more information.

The operation responds with a status message indicating whether the operation was successful.

Chapter 8. Managing The Infrastructure

8.1. 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.

8.1.1. Activity Collector

Object Name: overlord.rtgov.collector:name=ActivityCollector

The activity collector has the following configuration properties:

Property	Description
CollectionEnabled	A boolean property that can be used to enable or disable activity collection within the server.

8.1.2. Activity Logger

Object Name: overlord.rtgov.collector:name=ActivityLogger

This component uses a batching capability to enable the information to be sent to the Activity Server as efficiently as possible. This mechanism has the following configuration properties:

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:

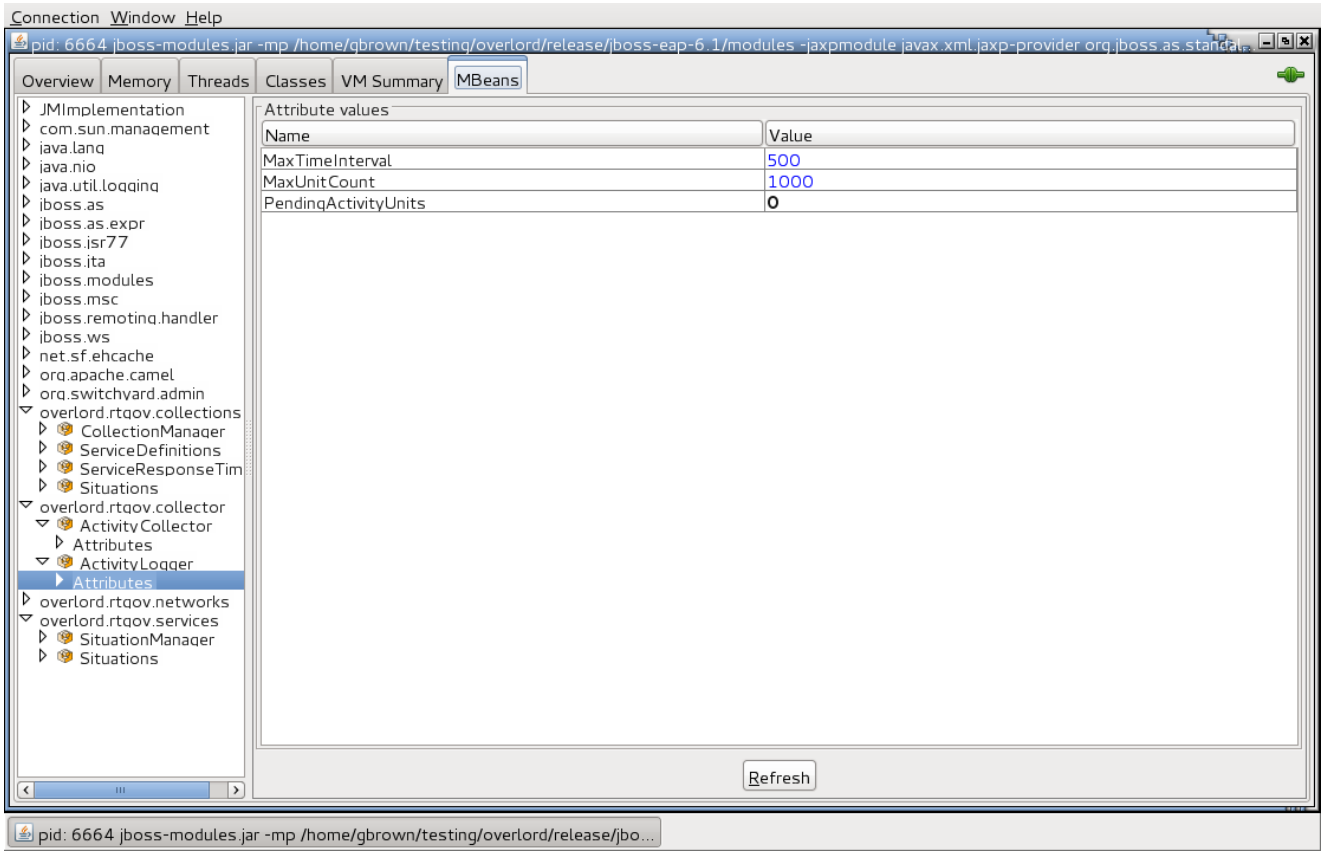

Figure 8.1.

The component also provides a *read-only* property:

Property	Description
PendingActivityUnits	This value indicates how many logger messages are waiting to be sent to the server. This can be used to guage how busy the collector is, and whether it is getting backed up.

8.2. 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.

8.2.1. Event Processor Network Manager

Object Name: overlord.rtgov.networks:name=EPNManager

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.

8.2.2. Event Processor Networks

Object Name: overlord.rtgov.networks:name=<name>,version=<version>

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:

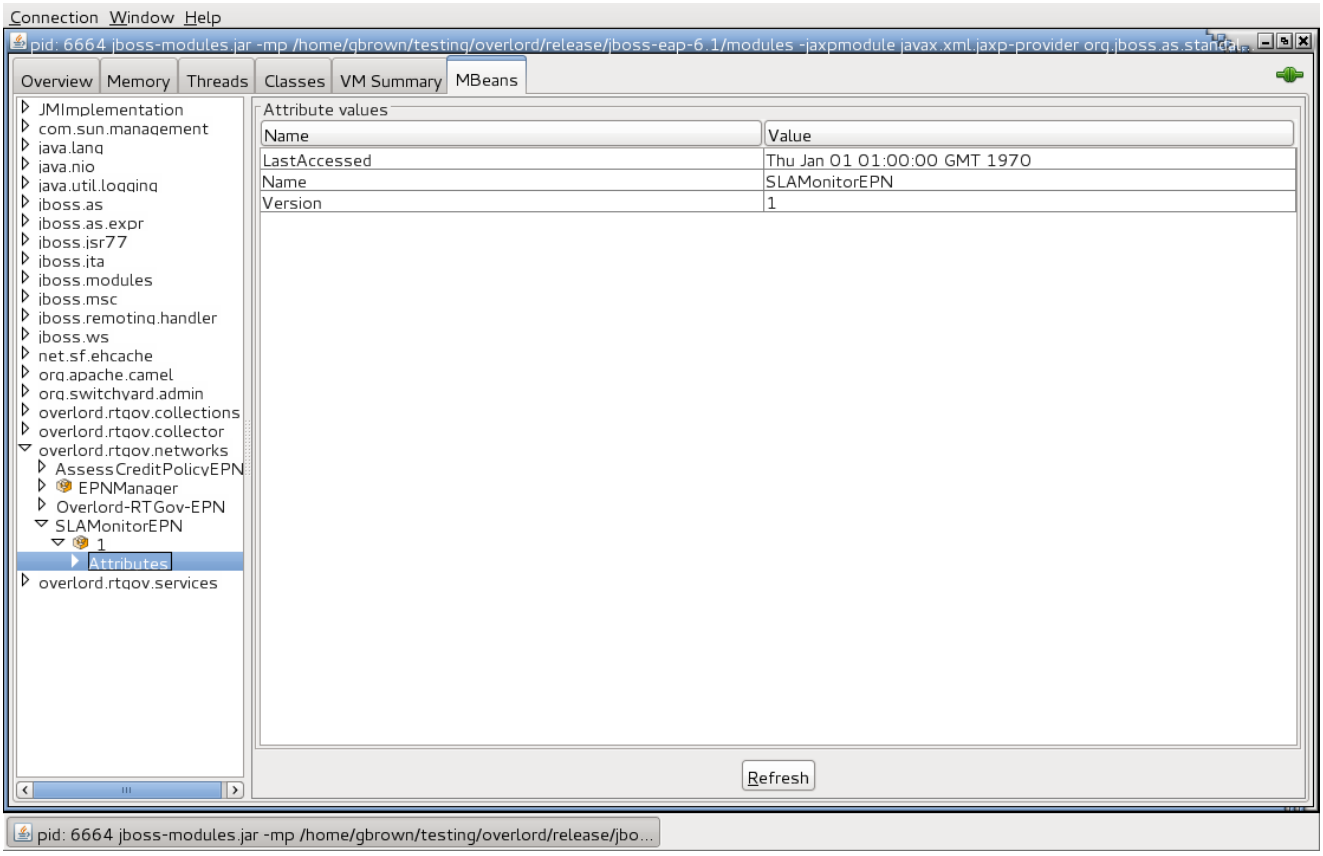

Figure 8.2.

8.3. 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.

8.3.1. Active Collection Manager

Object Name: overlord.rtgov.collections:name=CollectionManager

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

Attribute	Description
	number of elements in the collection has been reached.

8.3.2. Active Collections

Object Name: overlord.rtgov.collections:name=<ActiveCollectionSourceName>

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:

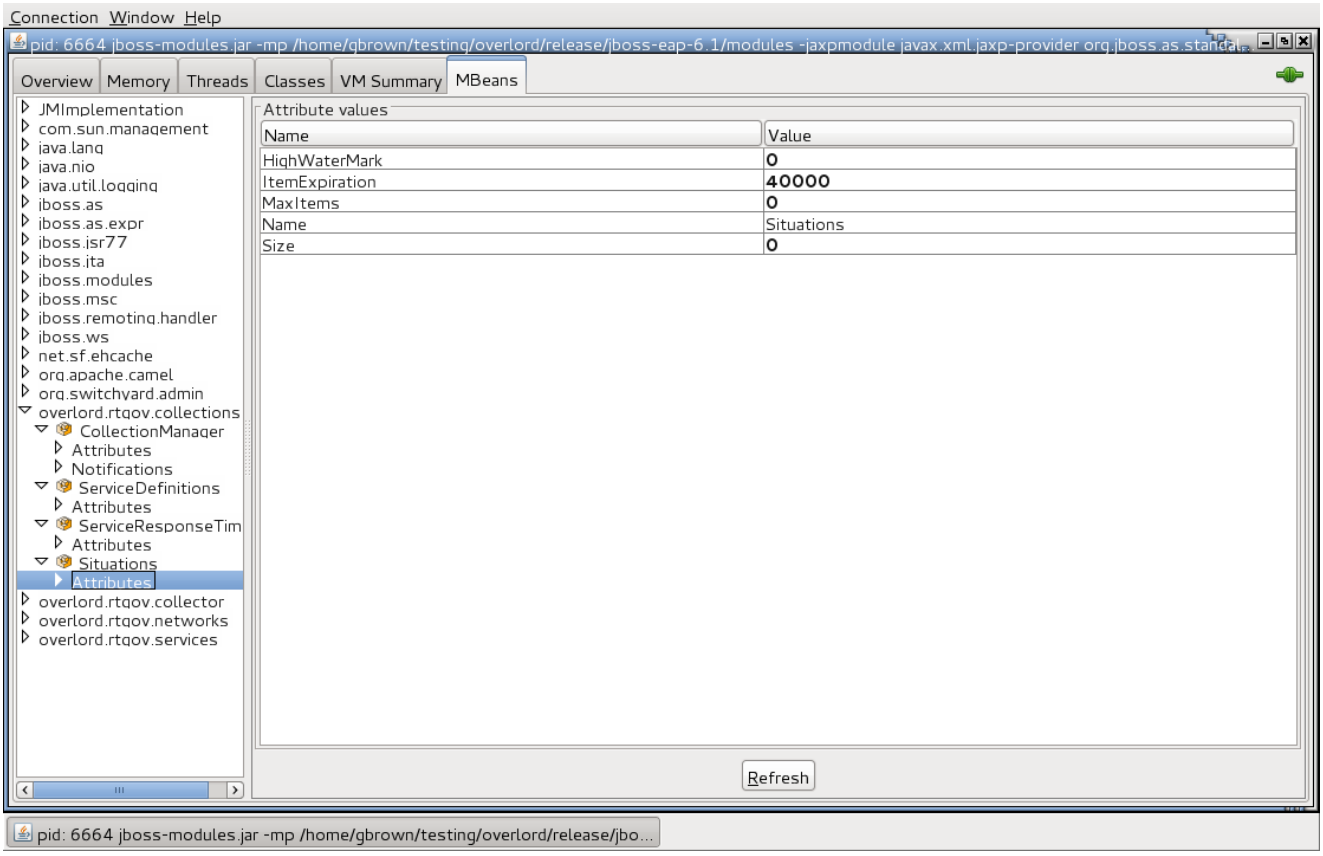
Connection Window Help
pid: 6664 jboss-modules.jar -mp /home/gbrown/testing/overlord/release/jboss-eap-6.1/modules -jaxpmodule javax.xml.jaxp-provider org.jboss.as.standal
Overview
Memory
Threads
Classes
VM Summary
MBeans
JMImplementation
com.sun.management
java.lang
java.nio
java.util.logging
jboss.as
jboss.as.expr
jboss.isr77
jboss.ita
jboss.modules
jboss.msc
jboss.remoting.handler
jboss.ws
net.sf.ehcache
org.apache.camel
org.switchyard.admin
overlord.rtgov.collections
CollectionManager
Attributes
Notifications
ServiceDefinitions
Attributes
ServiceResponseTim
Attributes
Situations
Attributes
overlord.rtgov.collector
overlord.rtgov.networks
overlord.rtgov.services
Attribute values
Name
Value
HighWaterMark
0
ItemExpiration
40000
MaxItems
0
Name
Situations
Size
0
Refresh
pid: 6664 jboss-modules.jar -mp /home/gbrown/testing/overlord/release/jbo...

Figure 8.3.