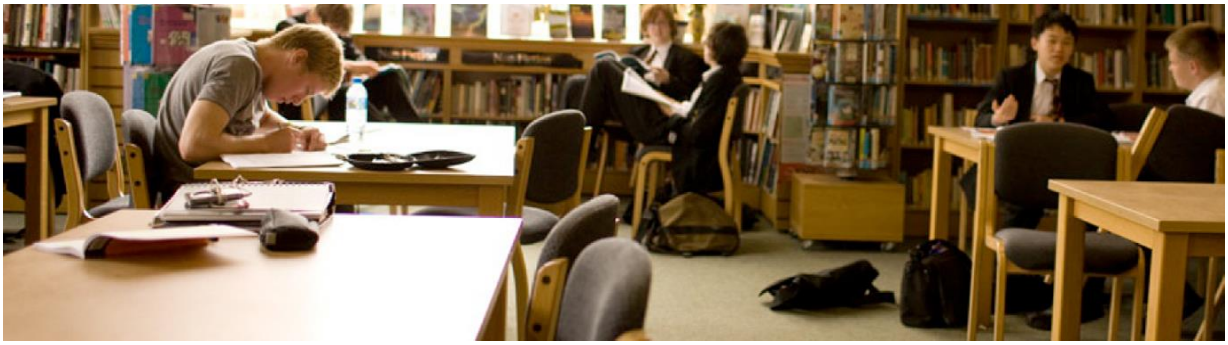
White Paper

Customizing Correlation Rules Best Practices Guide

Last Updated: October 7, 2015



Contents

Introduction

Cisco Prime™ Collaboration Assurance 10.5 and higher provides you with an extensible correlation engine. This engine can be used to correlate the events reported on your Cisco® collaboration network based on rules. Event correlation streamlines dependent events into fewer alarms, and the Cisco Prime Collaboration Assurance event correlation engine helps reduce the clutter of events and alerts on dashboards. Users can easily see key alerts and events on the alarm browser and can take action to fix them, improving mean time to repair (MTTR). Event correlation offers built-in rules to correlate data and generate aggregated alerts.

You can embed the correlation intelligence in the form of rules using pure SQL without the need to write any Java or C++ code.

Cisco Prime Collaboration Assurance offers three kinds of correlation rules - time-based correlation, threshold-based correlation, and root-cause correlation.

The following use cases describe how to implement the custom correlation rules using the correlation engine.

Use Case 1: Repeated Occurrence

Assume there are call failures in your collaboration network. Whenever a call routing through a particular route list (RL) fails due to resource exhaustion, a RouteListExhausted alarm is raised. If you do not wish to be notified on every failure, you can use the correlation engine to create a simple rule that raises an alarm when a definitive failure event occurs. For example, you can create a rule to raise an alarm when 10 failures occur in a span of five minutes.

You can create a rule using the correlation engine as follows:

1. Log in as globaladmin to the Cisco Prime Collaboration Assurance application, and navigate to **Administration > Alarm & Event Setup > Event Customization**.
2. Click **Add** in the Correlations Rules tab. In the popup window, specify:
   a. Rule name; for example, Repeated RL Failure Rule.
   b. Correlation alarm name that must be raised when the rule condition is satisfied; for example, Repeated RL Failure.
   c. Description for the alarm; for example, “This alarm would be raised when multiple RL failures happen within a short span of time”.
   d. Severity for the correlated alarm; for example, Critical.
   e. The rule:

```
select deviceid, source as componentname, array_accum(id) as ids from
cpcm_schema.allfaultevent_stream < VISIBLE '5 minutes' ADVANCE '1 minute' >
where name='RouteListExhausted' group by deviceid, source having count(*) >=
10;
```

3. Click **Save**.

How the Rule Is Applied: Example

Consider that there are four RouteListExhausted failures that occurred on a particular route list configured on Cisco Unified Communications Manager, at 10:00 a.m. No alarms are generated.

At 10:02 a.m., three more RouteListExhausted failures occur on the same route list. No alarms are generated.

Now, at 10:03 a.m., when there are three or more RouteListExhausted failures detected on the same route list, the correlation condition is met and thus an alarm named Repeated RL Failure (as specified while creating the rule) is generated in the system. This correlation alarm has 10 RouteListExhausted events (which occurred from 10:00 a.m. to 10:03 a.m.) tagged to it.

As a result, the rule reduces noise, raising just one significant alarm instead of raising multiple individual raw alarms.

Interpreting the Query

```
select
    deviceid,
    source as componentname,
    array_accum(id) as ids

from cpcm_schema.allfaultevent_stream
    <VISIBLE '5 minutes' ADVANCE '1 minute'>

where     name='RouteListExhausted'

group by deviceid, source

having count(*) >= 10;
```

- The `select` clause provides reference to the allfaultevent_stream, which is the fault stream. For details on the fault stream, see Appendix A.
  - The variables, `deviceid, source`, and `id`, refer to the attributes in the fault stream.
  - The variables `componentname` and `ids` are internally used and must not be renamed.
- The `from` clause provides reference to the fault stream and sliding window interval. You can change the fault stream window to the desired value.
- The `where` clause provides reference to the event name that matches the specified condition. The attribute `name` refers to the name of the event in the fault stream. For details, see Appendix A.
- The `group by` clause specifies an additional condition wherein the grouping occurs at device and component levels. The `deviceid` and `source` attributes refer to the device and the component on which the event was raised, respectively. You can group by any attribute available in the fault stream.
- The `having` clause specifies the frequency of occurrence. You can modify the value as desired.

You now can use this rule for any kind of repeated event matching - and customize it to suit your needs.

Use Case 2: Logical Implication (A → B)

Logical Implication is used for modeling correlation scenarios where one condition follows one or more other conditions. The truth table helps you determine whether or not a scenario belongs to this category. For simplicity, consider a logical implication involving two conditions - A and B.

Logical Implication is mathematically represented by A → B (read as A implies B).

A	B	Result (A → B)
T	T	T
T	F	F
F	T	T
F	F	T

A classic example from the unified communications world is the **CodeRed** scenario. When **CodeRed** conditions occur on the Cisco Unified Communications Manager, the Unified Communications Manager service is automatically restarted. Two alarms are generated on Cisco Prime Collaboration Assurance at different instances in a short interval - CodeRed and Service Down.

Using the truth table, first determine whether this scenario really falls under logical implication:

A (=CodeRed)	B (=Service Down)	Result
T	T	CodeRed occurs and Service Down occurs; this is a valid scenario. The result is T.
T	F	CodeRed occurs, but Service Down does not occur; this is an invalid scenario. The result is F.
F	T	CodeRed does not occur and Service Down occurs: this is a valid scenario since the service can go down because of other reasons. The result is T.
F	F	Neither CodeRed nor Service Down occurs: this is a valid scenario. The result is T.

This truth table matches with the A → B Truth Table, and therefore this scenario actually belongs to the logical implication category.

From a correlation perspective, for this scenario, a single alarm must be raised instead of two. In order to achieve this, you can use the correlation framework:

1. Log in as globaladmin to the Cisco Prime Collaboration Assurance application and navigate to **Administration > Alarm & Event Setup > Event Customization**.
2. Click **Add** in the Correlations Rules tab. In the popup window, specify:
   a. Rule name; for example, CodeRed Rule.
   b. Correlation alarm name that must be raised when the rule condition is satisfied; for example, CodeRed Detected.
   c. Description for the alarm; for example, “This rule correlates root cause and symptom alarms”.
   d. Severity for the correlated alarm; for example, Critical.

e. The rule:

```
select array_accum(fault_event.id) as ids, array_accum(fault_event.name)as
names,fault_event.deviceid as deviceid,cq_close(*)from
cpcm.allfaultevent_stream as fault_event <VISIBLE '5 minutes' ADVANCE '1
minute'> where fault_event.name ='CodeRed' or (fault_event.name ='ServiceDown'
and fault_event.source=”Cisco Call Manager”)group by deviceid having ( sum
(CASE WHEN name='CodeRed' then 1 ELSE 0 END)=1 );
```

3. Click **Save.**

How the Rule Is Applied: Example

Consider that a CodeRed event is generated on Cisco Unified Communications Management and an alarm is generated at 10:00 a.m.

At 10:01 a.m., the Unified Communication Manager service is restarted, and at 10:02 a.m. a ServiceDown event is generated.

The system will now apply the correlation rule to associate the ServiceDown event with the CodeRed alarm, instead of raising ServiceDown as a separate alarm.

Therefore, the rule reduces the noise, raising just one alarm (for the root cause) instead of raising alarms for all of the symptoms.

Interpreting the Query

```
select
   array_accum(fault_event.id) as ids,
   array_accum(fault_event.name) as names,
   fault_event.deviceid as deviceid

from cpcm.allfaultevent_stream as fault_event
   <VISIBLE '5 minutes' ADVANCE '1 minute'>

where   fault_event.name ='CodeRed'
   or (fault_event.name ='ServiceDown'
               and
       fault_event.source=”Cisco Call Manager”)

group by deviceid

having (sum(CASE WHEN name='CodeRed' THEN 1 ELSE 0
END)=1);
```

- The `select` clause provides reference to the allfaultevent_stream, which is the fault stream. For details on the fault stream, see Appendix A.
  - The variables, `deviceid`, `name`, and `id`, refer to the attributes in the fault stream.
  - The variables `names` and `ids` are internally used and must not be renamed.
- The `from` clause provides reference to the fault stream and sliding window interval. You can change the fault stream window to the desired value.

- The `where` clause provides reference to the event name that matches the specified condition. The attribute `name` refers to the name of the event in the fault stream, and the `source` attribute refers to the component name. For details, see Appendix A.
- The `group by` clause specifies an additional condition wherein the grouping occurs at device and component levels. The `deviceid` attribute refers to the device on which the event was raised. You can group by any attribute available in the fault stream.
- The `having` clause specifies an additional condition specific to the logical implication. This condition is met whenever there is exactly one CodeRed event over a given window and for a particular device.

You can create correlation rules for different conditions such as logical conjunction, logical disjunction, and so on in a similar way.

Use Case 3: Correlation Across Multiple Streams

Generally the events in the fault stream do not contain all of the details required for correlation. Hence, it is often necessary to refer to an external data source such as inventory, or equivalent, in order to perform the correlation. The Cisco Prime Collaboration correlation engine simplifies this task.

Using the correlation engine, you can create a view (see Appendix C) and use the view in your SQL query. In addition, Cisco Prime Collaboration Assurance contains a built-in set of views (see Appendix B), which can be readily used. These views currently are focused on Cisco Unified Contact Center deployment but can be easily extended to Cisco Unified Communications Manager, Cisco Unity® Connection, or any deployment.

As an example, let us consider a Cisco Unified Contact Center deployment, where an alarm must be raised only if both UCCE Router on side A and UCCE Router on side B are down.

You can use the correlation engine to create a rule accordingly:

1. Log in as globaladmin to the Cisco Prime Collaboration Assurance application, and navigate to **Administration > Alarm & Event Setup > Event Customization**.
2. Click **Add** in the Correlations Rules tab. In the popup window, specify:
   a. Rule name; for example, UCCE Router Down.
   b. Correlation alarm name that must be raised when the rule condition is satisfied; for example, UCCE Router Down.
   c. Description for the alarm; for example, “This alarm would be raised when router is down on both side A and side B”.
   d. Severity for the correlated alarm; for example, Critical.
   e. The rule:

```
select array_accum(deviceid) as deviceids, array_accum(source) as
componentnames, array_accum(id) as ids from cpcm_schema.allfaultevent_stream
as faultevent <VISIBLE '60 minutes' ADVANCE '1 minute'>,
cpcm_ccroutersidesview as ccroutersides where faultevent.name='ComponentDown'
and faultevent.source like '%Router%'
and ARRAY[faultevent.deviceid] <@ ccroutersides.idlist group by
array_to_string(ccroutersides.idlist, ',') having ((SUM(CASE WHEN
name='ComponentDown'THEN 1 ELSE 0 END) == 2) );
```

3. Click **Save**.

How the Rule Is Applied: Example

Consider that the Cisco Unified Contact Center Enterprise (UCCE) Router on side A goes down at 10:00 a.m., resulting in a ComponentDown event. An alarm is raised immediately as the side B router takes over.

At 10:30 a.m., if the UCCE Router on side B also goes down, resulting in another ComponentDown event, the correlation engine triggers a single alarm named UCCE Router Down and associates the two ComponentDown events to this alarm.

Therefore, the rule reduces noise by raising a single alarm for the service impact instead of raising several individual raw alarms.

Interpreting the Query

```
select array_accum(deviceid) as deviceids,
          array_accum(source) as componentnames,
          array_accum(id) as ids

from cpcm_schema.allfaultevent_stream as faultevent
<VISIBLE '60 minutes' ADVANCE '10 minute'>,
        cpcm_ccroutersidesview as ccroutersides

where faultevent.name='ComponentDown' and
faultevent.source like '%Router%' and
         ARRAY[faultevent.deviceid] <@
             ccroutersides.idlist

group by array_to_string(ccroutersides.idlist, ',')

having ((SUM(CASE WHEN name='ComponentDown' THEN 1
                           ELSE 0 END) == 2));
```

- The `select` clause provides reference to the allfaultevent_stream, which is the fault stream. For details on the fault stream, see Appendix A.
  - The variables, `deviceid, source`, and `id`, refer to the attributes in the fault stream.
  - The variables `componentnames` and `ids` are internally used and must not be renamed.
- The `from` clause provides reference to the fault stream and sliding window interval. You can change the fault stream window to the desired value. The `from` clause also specifies the inventory view cpcm_ccroutersidesview used in the query. For details on this inventory view, see Appendix B.
- The `where` clause provides reference to the event name that matches the specified condition. The attribute `name` refers to the name of the event in the fault stream. For details, see Appendix A.
- The `group by` clause specifies an additional condition wherein the grouping occurs at `idlist` and component levels. The `idlist` is a contact center inventory view attribute that contains an array of all router devices part of a UCCE duplex pair (combination of side A and side B).
- The `having` clause specifies an additional condition that exactly two ComponentDown events must occur on the router component, as there are two sides in a Cisco Unified Contact Center duplex pair.

Appendix A - Fault Stream Schema

All of the relevant Cisco Prime Collaboration Assurance rules are executed on the fault stream. This stream contains events that pertain to various rules as a common stream is maintained for all events and rules.

In order to write the rule you must understand the fault stream completely:

```
create stream allfaultevent_stream(
id bigint, alarmCode bigint, category varchar(200),
code bigint, componentName varchar(200),
description varchar(1000), deviceId bigint,
deviceName varchar(200), entityId bigint,
name varchar(200), severity integer,
source varchar(200), subcategory varchar(200),
subid varchar(200), component varchar(200), qtime timestamp cqtime system);
```

The following table describes fields in the fault stream:

Name	Data Type	Description
allfaultevent_stream	-	Table name (in your SQL rule) that refers to the fault stream
id	integer	Event ID of the event
alarmCode	integer	Alarm code of the event as defined in the catalog file
category	string	Indicates whether an event is an infrastructure event, Unified Communications Manager event, and so on
componentName	string	Unused
description	string	Detailed description of the event
deviceId	integer	The ID of the device on which this event is raised
entityId	integer	Unused
name	string	Name of the event
severity	integer	Severity of the event
source	string	Component on which the event is raised
subcategory	string	Unused
subid	string	Unused
component	string	Holds event-specific data (varies from event to event)
qtime	timestamp	The time at which the event occurred

Appendix B - Unified Contact Center Deployment Views

The Cisco Unified Contact Center deployment inventory views available in Cisco Prime Collaboration Assurance are as follows. These views can be used based on the correlation use cases you have in your deployment. To create your own view, see Appendix C.

cpcm_cmdevview

View Definition

```
create or replace view cpcm_cmdevview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select id,devicetype from cmdevice')
as t(id bigint ,devicetype text));
```

Output

```
         id    |       devicetype
--------+-------------------------
3819822 | ContactCenterEnterprise
3819854 | ContactCenterEnterprise
3819853 | ContactCenterEnterprise
3819831 | CIM
3819847 | Unknown
3819851 | CustomerVoicePortal
3819855 | CustomerVoicePortal
3819821 | CUCM
3819852 | CustomerVoicePortal
3819819 | SOCIALMINER
3819838 | FINESSE
3819820 | Unknown
3819828 | CUIC
3819840 | CUCM
3819834 | Unknown
3819835 | MEDIASENSE
3819829 | Unknown
3819830 | CUIC
3819836 | CUCM
3819846 | CustomerVoicePortal
3819823 | ContactCenterEnterprise
```

cpcm_ccvpview

View Definition

```
create or replace view cpcm_cccvpview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select ARRAY[ccpim.deviceid,cmdev1.id]
from contactcenterpim as ccpim,cmdevice as cmdev1 where
cmdev1.ipaddress=ccpim.cccapimperipheralhostname and
ccpim.cccapimperipheraltype=''vru'';') as t(idlist bigint[]));
```

Output

```
 idlist
-----------------
{3819853,3819855}
{3819822,3819855}
{3819853,3819846}
{3819822,3819846}
{3819853,3819845}
{3819822,3819845}
{3819853,3819856}
{3819822,3819856}
```

cpcm_ccmediaview

View Definition

```
create or replace view cpcm_ccmediaview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select ARRAY[ccpim.deviceid,cmdev1.id]
from contactcenterpim as ccpim,cmdevice as cmdev1 where
cmdev1.ipaddress=ccpim.cccapimperipheralhostname and
ccpim.cccapimperipheraltype=''mediaRouting'';') as t(idlist bigint[]));
```

Output

```
idlist
-----------------
{3819822,3819822}
{3819822,3819853}
{3819822,3819819}
{3819822,3819834}
{3819822,3819848}
{3819822,3819850}
```

cpcm_ccrouterpgview

View Definition

```
create or replace view cpcm_ccrouterpgview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select ARRAY
[ccrouter.deviceid,ccpg.deviceid] from contactcenterrouter as
ccrouter,contactcenterpg as ccpg where (ccpg.cccapgroutersideaname IN (select
LOWER(cccarouterpublichighaddr) from contactcenterrouter)) or
(ccpg.cccapgroutersidebname IN (select LOWER(cccarouterpublichighaddr) from
contactcenterrouter)) group by ccrouter.deviceid,ccpg.deviceid;') as t(idlist
bigint[]));
```

Output

```
idlist
-----------------
{3819822,3819822}
{3819822,3819853}
{3819853,3819822}
{3819853,3819853}
```

cpcm_ccpgsidesview

View Definition

```
create or replace view cpcm_ccpgsidesview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select
ARRAY[ccpg.deviceid,cc.deviceid] from contactcenterpg as ccpg,contactcenter as cc
where ccpg.cccapgside=''sideA'' and
upper(ccpg.cccapgduplexpairname)=upper(cc.cccaname) and (''sideB'' IN (select
cccapgside from contactcenterpg where contactcenterpg.deviceid=cc.deviceid));')
as t(idlist bigint[]));
```

Output

```
idlist
-----------------
{3819822,3819853}
```

cpcm_ccrouterssidesview

View Definition

```
create or replace view cpcm_ccroutersidesview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select
ARRAY[ccrouter.deviceid,cc.deviceid] from contactcenterrouter as
ccrouter,contactcenter as cc where ccrouter.cccarouterside=''sideA'' and
upper(ccrouter.cccarouterduplexpairname)=upper(cc.cccaname) and (''sideB'' IN
(select cccarouterside from contactcenterrouter where
contactcenterrouter.deviceid=cc.deviceid));') as t(idlist bigint[]));
```

Output

```
idlist
-----------------
{3819822,3819853}
```

cpcm_ccrouterloggerview

View definition

```
create or replace view cpcm_ccrouterloggerview as (select * from
dblink('port=5433 dbname=cpcm user=cmuser password=cmuser', 'select ARRAY
[ccrouter.deviceid,cclogger.deviceid] from contactcenterrouter as
ccrouter,contactcenterlogger as cclogger where
(cclogger.cccaloggerroutersidebname IN (select cccarouterpublichighaddr from
contactcenterrouter)) or (cclogger.cccaloggerroutersideaname IN (select
cccarouterpublichighaddr from contactcenterrouter));') as t(idlist bigint[]));
```

Output

```
idlist
------------------
{3819853,3819854}
{3819822,3819854}
{3819853,3819823}
{3819822,3819823}
```

cpcm_ccloggersidesview

View definition

```
create or replace view cpcm_ccloggersidesview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select
ARRAY[cclogger.deviceid,cccomponent.deviceid] from contactcenterlogger as
cclogger, contactcentercomponent as cccomponent where
cclogger.cccaloggerside=''sideA'' and cccomponent.cccacomponentname=''LoggerB''
and cclogger.cccaloggerroutersideaname=(select cccaloggerroutersideaname
```

Output

```
idlist
-----------------
{3819854,3819823}
```

cpcm_ccucmview

View definition

```
create or replace view cpcm_ccucmview as (select * from dblink('port=5433
dbname=cpcm user=cmuser password=cmuser', 'select
ARRAY[ccpim.deviceid,cmdev1.id,ccpim1.deviceid] from contactcenterpim as
ccpim,cmdevice as cmdev1,contactcenterpim as ccpim1 where
cmdev1.ipaddress=ccpim.cccapimperipheralhostname and
cmdev1.ipaddress=ccpim1.cccapimperipheralhostname and
ccpim.cccapimperipheraltype=''callManager'';') as t(idlist bigint[]));
```

Output

```
idlist
-------------------------
{3819853,3819821,3819822}
{3819853,3819821,3819822}
{3819853,3819821,3819853}
{3819822,3819821,3819822}
{3819822,3819821,3819822}
{3819822,3819821,3819853}
```

Appendix C - Creating and Editing User-Defined Views

To create a view:

1. Log in as root to the Cisco Prime Collaboration Assurance server.
2. Connect to the Truviso database and create a view using the following commands:

```
#cd /opt/pa/TruCQ/bin
#./psql -p 5435 -d cqdb -U primea
#cqdb=# execute the create view statement here
#cqdb=# \q
```

To edit a view:

1. Log in as root to the Cisco Prime Collaboration Assurance server.
2. Enter `#goemsam`.
3. Stop the Cisco Prime Collaboration Assurance services using the command:

```
#bin/cpcmcontrol.sh stop
```

4. Edit the view using the following commands:

```
#bin/start_trucq.sh
#cd /opt/pa/TruCQ/bin
#./psql -p 5435 -d cqdb -U primea
#cqdb=# execute the update view statement here
#cqdb=# \q
#goemsam
#bin/shutdown_trucq.sh
```

5. Start the Cisco Prime Collaboration Assurance services using the command:

```
# bin/cpcmcontrol.sh start
```

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Printed in USA C11-732182-01 10/15