The Event Management Process

This chapter details the event correlation methods, the second of the AOC extensions from Cisco Mobile Wireless Fault Mediator. It examines the complete structure of an event correlation method and details each component and the attributes used to construct a method. Finally, there is a preview of the role played by the method conclusions, the final stage of the fault management process.

Managing events

The goal of event management within Cisco Mobile Wireless Fault Mediator is to reduce the amount of superfluous network status information presented to the user. This takes the form of correlating certain events in the event stream together to arrive at a single root cause, suppressing events, changing their severity under certain circumstances, deleting events that are no longer relevant, or even suspending polling to particular devices when certain events are received from them.

This process involves several key steps: storing events, filtering events, analyzing relationships between events on the network and the network topology and, finally, taking action based on the correlation process. Each of these areas is now discussed before introducing the event correlation methods that define how AMOS, the correlation engine of Cisco Mobile Wireless Fault Mediator, functions.

Event storage

Each event that transpires on the network must be logged or stored somewhere for later use and for AMOS this is the AMOS Events database. The AMOS Events database contains the current set of events and alerts. Each record contains a series of column names which are used by AMOS to determine what action needs to be undertaken on each event. The event records are well-structured to ensure event records are handled consistently. The column names of an event record and the AMOS Events database are thoroughly explored in Appendix A, “AMOS Databases.”
Event filtering

The second step is to determine whether an incoming event satisfies a conditional test. The nature of the test is described in further detail in the section on the event correlation methods later in this chapter.

Correlation and topology considerations

For all events that pass the filtering process, the third step is to undertake a correlation process. This process involves comparing the column names of the incoming event to the network topology model, Containment Model and other events that have transpired on the network to establish whether there is a relationship between events; for instance, did the failure of one device cause a series of events on devices “downstream” of the failed device? Or are there other implications for devices connected to the failed device?

Taking action

The final step is to action any faults that have been identified as a result of the correlation stage. Should events on other network devices be suppressed or deleted as a consequence of an event? What database modifications inserts, updates and deletes are required as a consequence or correlation? Are directives required to be run due to correlated results?

The event correlation methods

The operation of AMOS is determined by the event correlation methods, which are extensions in the AOCs; thus, each instance of an object instantly has event correlation methods associated with it. When AMOS is launched, it extracts (via the MWFM NMOS component CLASS) the event correlation methods from the relevant AOCs. The event correlation methods are then on standby until initiated by a trigger. The triggers are one of the four components of the Method Template:

1. The method name.
2. The method triggers.
3. The method firing condition.
4. The method conclusion.

Each event correlation method follows a standard template to fulfill the requirements outlined in the “Managing events” section on page 5-1.

Each method’s template is constructed in the AOC browser (see Figure 3-2 on page 3-4).
The method name

The method name is an attribute that declares the name of the current method. Any string of text is accepted as a valid entry and must be unique within the specified AOC. For example, “Create” and “Escalate” are valid method names.

A feature of both the event correlation methods and the poll definitions is the economy of description offered by the AOCs’ inheritance of policies. Classes lower in the hierarchy will inherit the functionality of all methods from their parent classes, unless the event correlation method is locally redefined to override the function of the inherited one. This is done by creating a new method in the child class with the same method name as the one derived from the parent class.

The method name is also used as a means of method control; it is the method name which is the basis for method chaining. For further information see the “Method chaining” section on page 5-6.
The method triggers

An event correlation method can be initiated or “triggered” in any of three ways:

1. By events in the event stream broadcast from MONITOR—a non-timed method.
2. By a timer defined in the event correlation method itself—a timed method.
3. As a result of applying method chaining—a non-timed method. The method can be triggered by events output from another method or by another method firing (or even after it not firing).

The notion of triggering introduces two associated terms hinted at above, timed and non-timed methods. Both are components of the method firing condition.

The method firing condition

The method firing condition is quite simply a series of conditional tests to determine whether the method conclusion is fired or not. If the conditional tests of the method firing condition are passed, then the method will be fired. In such a case, processing will continue and the method conclusion will be run (activated). If the tests in the method firing conditions are not passed, processing is terminated. The method is not fired and the method conclusion is not run.

Note

Firing, in the context of an event correlation method, is when the method firing condition is successfully evaluated, and processing continues to the method conclusion, where some action takes place.

The firing condition can be broken down into four key components:

• The method timing section
• The method filtering section
• The method chaining section
• The firing policy

These components are discussed below.

The method timing section

The first component of the firing condition relates to the triggers outlined in the previous section. As specified, there are three triggers. These triggers, and thus the operation of the event correlation methods, can be further categorized into timed and non-timed methods.

Timed event correlation methods

A timed method is one that is triggered by specifying a time in seconds, minutes, hours or days in the relevant field in the AOC browser. For example, specifying a time of five minutes will mean the event correlation method will be triggered every five minutes and run on any event present in the AMOS Events database that satisfies the remainder of the conditional tests in the method firing condition.
Non-timed event correlation methods

A non-timed method is one that is triggered on the basis of an interrupt from an external source. There are two external sources:

1. Events in the event stream broadcast from MONITOR.
2. As a result of applying method chaining. In such cases the source is events output from another method or events in the AMOS Events database.

In both cases, the conditional tests in the method firing condition will be run on the event that triggered the event correlation method, whether that is the event(s) in the event stream or the event(s) output from another method.

Figure 5-2 The three triggers’ input sources

Event and entity filtering

The next component of the method firing condition is the conditional test. Cisco Mobile Wireless Fault Mediator uses two means of filtering in the firing condition, the event filter and the entity filter. The event filter allows you to filter on the column names of an event record contained within the AMOS Events database, while the entity filter allows you to set criteria based on column names of entities in the AMOS Entity database.
The event filter

The event filter uses a series of logical operators and the column names of the AMOS Events database to constrain the execution of an event correlation method. For example, you may choose to define a method called “routerPingFail” which runs on ping fail events from routers when the Severity of the event is greater than “Unknown” (e.g., Minor, Major, Critical - see Table A-1 on page A-1, for the precedence of the values in the Severity column name of an event record). The event filter for such a condition would be defined as shown in Table 5-1.

Table 5-1  Specifying the event filter

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>event filter</td>
<td>&quot;EventName='pingFail' AND Severity &gt; eval(int,'$UNKNOWN')&quot;</td>
</tr>
</tbody>
</table>

You can see how the column names of an event record can be used in a filter as well as some OQL syntax, namely the eval statement. EventName and Severity are column names of an event record (as tabulated and described in Appendix A, “AMOS Databases.” The eval statement enables the column names of an event record to be accessed and evaluated. The syntax and functionality of the eval statement is comprehensively explored in Appendix B, “The Eval Statement.”

In the example, the event filter portion of the firing condition would be passed if the incoming event record was a pingFail with a Severity of greater than Unknown.

The entity filter

The second filtering condition, the entity filter, operates in the same manner as the event filter except it filters against the network topology constraining the execution of an event correlation method using column names in the AMOS Entity database and a series of logical operators. The column names are outlined in Table A-2 on page A-3.

For instance, you may choose to limit the method to run only on events from a particular part of the network or subnet. In this case the entity filter would be defined as shown in Table 5-2.

Table 5-2  Specifying the event filter

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity filter</td>
<td>&quot;Address = eval(list type text, this --&gt; Address)&quot;</td>
</tr>
</tbody>
</table>

Method chaining

Method chaining allows a series of simple methods to be combined to form more complex methods; output from one method can be utilized and processed by a second method after the first has carried out its processing. This improves efficiency in determining root causes and streamlines the number of methods required in the AOCs.

Method chaining is achieved by linking event correlation methods via the Event Method dialog in the AOC browser. The process of linking methods in the Event Method dialog injects a series of entries into the method firing condition of the event correlation methods. These entries not only link the event correlation methods but also determine the sequencing of the methods; there are four method chaining possibilities. These are described in Table 5-3.
The firing policy

The firing policy is the final attribute that dictates whether the event correlation method conclusion will be fired or not. The firing policy is instigated after the event and entity filtering and method chaining has been assessed. There are three possible attributes available when specifying the firing policy:

- **Time-to-live (Ttl)**—an integer value that determines how many times the method is fired (or “actioned”) before it is deactivated. For example, if the Ttl value is “3” the method will fire three times and then deactivate until an external process (such as input from the AOC browser) reactivates it. If a value of zero is specified the method will fire indefinitely.

- **Number of fires before execute**—an integer value that specifies the number of times the method firing condition must be evaluated successfully before the method conclusion is executed. If the specified integer has a value of “4”, this means the method firing condition must be satisfied four times; on the fourth time, the method conclusion will run. Furthermore, once the method firing condition has been evaluated the number of times specified by the “number of fires before execute” attribute (in this case four), the method will be fired on each subsequent successful evaluation of the method firing condition. Thus, in this example, the method will be fired on the fifth successful evaluation, the sixth successful evaluation, etc. If a value is omitted or specified as zero, the method always fires.

- **Repeat condition**—an integer value that operates similar to the number of fires before execute condition, except it repeats after the method conclusion has been executed once. If the integer specified is “n”, then the method conclusion will be fired after the n, 2n, 3n, etc. successful evaluations of the method firing condition. For instance, if a value of “3” is specified the method conclusion will be run after the third successful evaluation. The method conclusion will not be run again until the method firing condition has been successfully evaluated on three more occasions. Thus, the method will be run on the sixth successful evaluation, the ninth successful evaluation, etc.

---

### Table 5-3 The four method chaining possibilities

<table>
<thead>
<tr>
<th>Chaining method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate after</td>
<td>Run the present method on the AMOS Events database after a previous method has been evaluated, regardless of whether it fires or not.</td>
</tr>
<tr>
<td>Evaluate after fired</td>
<td>Run the present method on the AMOS Events database after a previous method fires.</td>
</tr>
<tr>
<td>Evaluate after not fired</td>
<td>Run the present method on the AMOS Events database after a previous method does not fire.</td>
</tr>
<tr>
<td>Evaluate when fired</td>
<td>Run the present method on events output from another method when that method fires.</td>
</tr>
</tbody>
</table>
The method conclusion

Once the method firing condition has been passed, processing of the method conclusion commences. The method conclusion consists of two components:

- The actions stage—determines how the event records in the AMOS Events database are manipulated; for example, inserting, updating and deleting event records.
- The method control daemons—these enable the actions in the method conclusions to be run on devices specified by the control location without having to re-run the entire event correlation method.

Similar to the event correlation methods, each component of the method conclusion requires a trigger or a mechanism to initiate it. These are now discussed with respect to the event management process; you should move to Chapter 6, “The Method Conclusion In Action,” for a detailed exploration of the method conclusion in action.

Constructing actions triggers

Action triggers are a list of all events and alerts that have transpired on all devices (and entities contained within the devices). The actions triggers are effectively the input to the actions stage of an event correlation method conclusion.

The sole purpose of developing actions triggers is to generate a more comprehensive event and alert list which will be the input to the actions stage of the event correlation method conclusion. Should events on all devices be considered? Or just events on instances of devices? Is it necessary to consider events on all devices connected to a particular device? Or events on devices isolated by the failure of a discrete entity? Should we consider any or all of the events the entities contained in one of the devices just described? These are complex questions; however, it is possible to construct methods to cover any of the situations identified.

In order to construct the actions triggers it is necessary to establish exactly which entities (and therefore, which events from these entities) are to be used for the actions stage.

To facilitate this, the network topology model and the Containment Model described in Chapter 2, “Object-oriented Principles and MWFM NMOS,” are used to enable events that transpire on entities contained within devices to be considered in the actions stage, even though these entities may not be instances of AOCs. The ability of Cisco Mobile Wireless Fault Mediator to combine the network topology model and the Containment Model is one of the keys to Cisco Mobile Wireless Fault Mediator’s ability to accurately establish the root cause of network problems.
Which devices do I run the event correlation method conclusion on?

The control location attribute is used to scope the method to particular network objects within the network topology model.

**Note**

Scoping a method is determining which network objects in the topology model will have the event correlation method conclusion applied to them.

There are three control location possibilities, as described Table 5-4 with reference to the sample network in Figure 5-3:

**Table 5-4  Control location possibilities**

<table>
<thead>
<tr>
<th>Control location</th>
<th>Assigned value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>0</td>
<td>Selects only the object that generated the event which triggered the event method. In Figure 5-4, the instance that generated the original event is object E. The actions triggers will consist of all events that have transpired on that instance (subject to the application of the Containment Model—described in the next section).</td>
</tr>
<tr>
<td>Isolated</td>
<td>1</td>
<td>Selects all objects that can only be reached by going though the object which generated the event that triggered the event correlation method. In Figure 5-4, the objects that would be isolated by the failure of Object E are Objects G, I, J and K. H can still be reached via F. The actions triggers will consist of all events on isolated objects plus the method triggering event (subject to the application of the Containment Model—described in the next section).</td>
</tr>
<tr>
<td>Connected</td>
<td>2</td>
<td>Selects all objects that are directly connected to the object which generated the event that triggered the event method, Object E. In Figure 5-4, these objects are B, F, G, and H. The actions triggers will consist of all events on connected objects plus the method triggering event (subject to the application of the Containment Model—described in the next section).</td>
</tr>
</tbody>
</table>
When a control location of isolated is specified, there is an additional requirement to relate the location of the Polling Agent to the network topology, called isolated from. This allows you to establish a direction or a relationship between the object where the event transpired in the network and where the object was polled from. In the sample network above, the polling location is Object A. We have previously identified the fact that the failure of Object E would cause a situation where Objects G, I, J and K are isolated from the network. This observation is only valid if Object A is the polling location. If the polling location was Object G, the entities isolated by the failure of Object E would be entirely different; they would be Objects A, B, C, D, F and H. Thus, it should be clear that when you want to identify certain objects that are isolated by the failure of another object you must clarify this by establishing a relationship between the object being polled and the polling location.
Specifying the polling location is achieved by extracting the IP address of the Polling Agent from the method triggering event. This can be done using an `eval` statement similar to the one shown in Table 5-5.

### Table 5-5 Specifying the Polling Agent location

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isolated from</code></td>
<td><code>&quot;EntityName = eval(text,'&amp;AgentAddress')&quot;</code></td>
</tr>
</tbody>
</table>

In the context of our original example, the above statement would have extracted the IP address from the polling location and equated it with Object A.

**What level of containment do I consider?**

Once you have established which objects (or devices) the event correlation method will consider events on, it is also necessary to specify the containment level you wish to allow, since objects can contain other entities. For instance, a switch has a chassis that contains a series of cards, which may in turn contain a series of ports; additionally, these ports may be associated with a series of VLANs. Specifying the target level of containment you wish to consider gives Cisco Mobile Wireless Fault Mediator information about how it is to recurse through the Containment Model resolved during the discovery process. The level of containment is established by using the `run in container` attribute which contains the `target` attribute and three additional arguments: `traverseUp`, `traverseDown` and `controlFlag`.

### Table 5-6 Establishing the level of containment

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>run in container</code></td>
<td><code>target = (traverseUp, traverseDown, controlFlag)</code></td>
</tr>
</tbody>
</table>

- **target**—a list of all entities for inclusion in the generation of the actions triggers; events transpiring on the entities in the target list will be included in the actions triggers. The list is based on the three integer fields (`traverseUp`, `traverseDown` and `controlFlag`) which designate the containment level. A description of each integer follows.
- **traverseUp**—an integer value used to specify the number of levels to traverse up through the Containment Model. For instance, if an event associated with a card in a switch has occurred, it is possible to recurse up one level of the physical container and capture all events that have transpired on the switch. These events can then be incorporated in the actions triggers.
- **traverseDown**—an integer value used to specify the number of levels to traverse down through the Containment Model. Consider the same card in the switch above; it is possible to traverse down the containment model to port level which will allow you to capture all events on the ports of the card that have transpired on the ports of the card. When traversing down through the Containment Model, all paths are considered; for instance, when traversing from card level to port level it is not possible to discriminate between events occurring on different ports. Any event occurring on any port on the card in question will be included in the actions triggers.
- **controlFlag**—used to denote whether the event(s) on the entities encountered when traversing up and down the levels of the Containment Model should be included or excluded when compiling the “new” list of events and alerts which are the actions triggers for the actions stage. If controlFlag is set to `include_recurse`, then any event transpiring on an entity encountered when traversing up or down the Containment Model will be included in the actions triggers. If controlFlag is set to...
excluderecursed, then any event transpiring on an entity encountered when traversing up and down the Containment Model will be excluded from the actions triggers; the only events considered will be those on the final destination entities.

Figure 5-4 demonstrates the uses of the run in containers command to recurse through the Containment Model. To disable recursing through the Containment Model the run in container attribute is simply left unassigned.

*Figure 5-4 Recursing through containers*

The output generated by applying the control location to the network topology model and considering the Containment Model is a list of EntityNames whose events and alerts will be considered in the actions stage of the method conclusions—these are the actions triggers.

The operation of the actions stage of the event correlation method conclusion is detailed in Chapter 6, “The Method Conclusion In Action.”
Initiating the method control virtual daemons

The method control virtual daemons will be initiated when the event correlation method is fired, i.e., the method firing condition is passed. However, to operate successfully there is an additional requirement; a location in the network topology model (specified by using the control location attribute) must be specified. Additionally, a containment level in the Containment Model can be specified by using the run in container attribute. This establishes the entities the method control virtual daemons will run on.

The operation of the method control virtual daemons is detailed in Chapter 6, “The Method Conclusion In Action.”

A final word on the method conclusion components

Before embarking on Chapter 6, “The Method Conclusion In Action,” and discussing the method conclusion in action, the following summarizes how the two stages are initiated:

1. The actions stage is initiated by the construction of the action triggers which are all events on the entities identified by applying the control location to the network topology model and specifying a containment level.
2. The method control virtual daemons are run on all entities identified by applying the control location to the network topology model and specifying a containment level.

Conclusion

This chapter has introduced the fundamentals behind event correlation and processing: event storage, filtering, correlation and action. These aspects form the foundation of the method template, which is the basis of all event correlation methods (the second of the AOC extensions) in Cisco Mobile Wireless Fault Mediator. This chapter has thoroughly explored the first three components of an event method: the method name, which is the basis of method chaining; the method firing condition, which determines when the method will be fired; and the method triggers, which determine how the final stage of an event correlation method, the method conclusion, is initiated.

The components of a method conclusion—the actions stage and the method control virtual daemons—determine how the AMOS Events database is updated and whether external scripts can be launched as a result of the event correlation process. The following chapter examines all the attributes of the method conclusions and demonstrates how they can be applied.