HP OpenView Event Correlation Services

When a fault occurs in a telecommunications system, it can cause an event storm of several hundred events per second for tens of seconds. HP OpenView Event Correlation Services (ECS) helps operators interpret such storms. It consists of an ECS Designer for the interactive development of correlation rules and an ECS engine for execution of these rules.

by Kenneth R. Sheers

Modern telecommunication technologies such as SDH/SONET (Synchronous Digital Hierarchy/Synchronous Optical Network) and ATM (Asynchronous Transfer Mode) can generate large numbers of events when a fault occurs. Every logical and physical element involved in delivering a service that uses the failed or faulty element can generate multiple events. This can result in an event storm of several hundred events per second for tens of seconds. The task of telecommunications operations staff is to determine the underlying cause from the thousands of events presented to them.

HP OpenView Event Correlation Services (ECS) is designed to deal with the problems associated with event storms in the telecommunications environment. The theory from which HP OpenView ECS technology has evolved was developed by HP Laboratories in Bristol, England.[1] ECS is delivered as two distinct components: the ECS engine and the ECS Designer.

The ECS engine is a run-time correlation engine. It executes a set of downloaded correlation rules that control the processing of event streams. At first release, the ECS correlation engine is integrated into the HP OpenView Distributed Management (DM) postmaster.

The ECS Designer is a graphical user interface (GUI) development environment that allows the correlation rules to be developed interactively by selecting, connecting, and configuring logical processing blocks. Once the rules have been created, their operation can be simulated and visualized using a log of events input to a correlation engine attached to the ECS Designer, with the engine's concept of time controlled by the ECS Designer.

The correlation rules are created using a visual paradigm of nodes connected together to form a correlation circuit. Events logically enter nodes via input ports and leave via output ports. An output port of one node is connected to an input port of another node in the circuit, so that events flow through the circuit from one node to the next. The functional node types provided with ECS are a superset of the basic types considered necessary and sufficient to perform real-time event correlation.

An event correlation circuit, Fig. 1, constitutes a set of correlation rules that can be compiled and downloaded to an event correlation engine. It consists of a series of interconnected and appropriately configured nodes, together with any associated data and relationship information. Fig. 2 shows the ECS architecture and the relationship of the correlation circuit to the ECS Designer and the ECS engine.

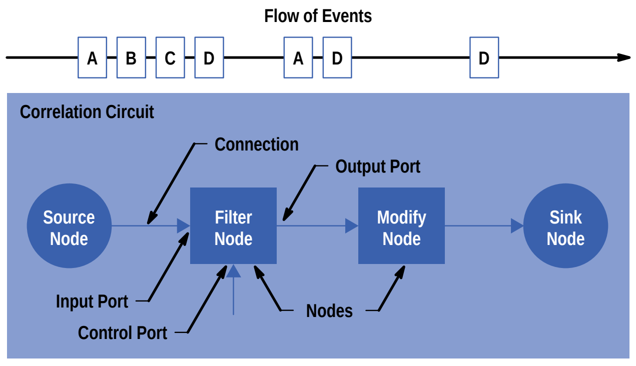


Fig. 1. *Generic correlation circuit. Correlation rules are specified by such circuits.*

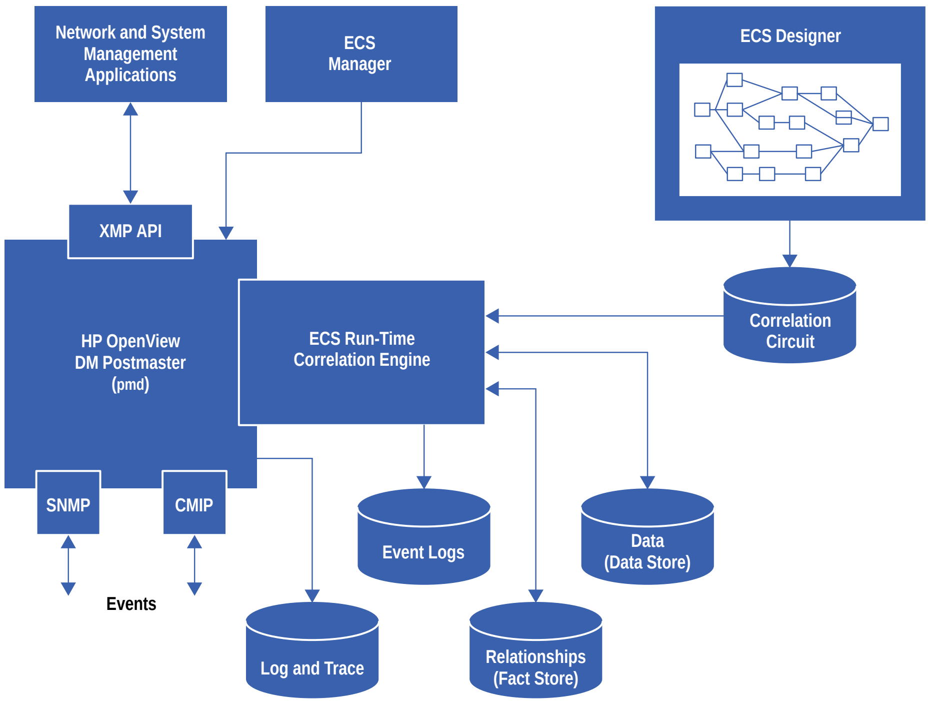

Network and System Management Applications
ECS Manager
ECS Designer
XMP API
HP OpenView DM Postmaster (pmd)
ECS Run-Time Correlation Engine
Correlation Circuit
SNMP
CMIP
Events
Event Logs
Data (Data Store)
Log and Trace
Relationships (Fact Store)

Fig. 2. *Architecture of HP OpenView Event Correlation Services.*

Real-Time Engine

A key differentiator of ECS compared with other correlation systems is that it operates in real time while taking into account the real-world problem that events will often be delayed in the management network and delivered to the correlation system out of order.

If events always arrived in bursts, it would be possible to buffer them on receipt and perform the correlation between bursts. However, event storms may be continuous, and the correlation engine should be capable of receiving the events, decoding them, and correlating them at the speed at which they arrive, continuously, without needing a mainframe to do the work. In ECS, any buffering required for correlation of time-separated events is dynamic and completely integrated into the normal engine operation.

In an ideal environment, a correlation engine is embedded into each piece of equipment that generates events. Correlated events from each piece of equipment are forwarded to another correlation engine where correlation across multiple systems is performed. This strategy reduces event volumes as early as possible, reducing the network traffic significantly. Thousands of events can be reduced to only a few events at each level of correlation.

Delayed Events

If all events were delayed by an equal amount within the management network, the correlation would simply be some delta time offset from real time. In a management network stressed by an event storm, some events will be delayed more than others. Consider a scenario in which an event B is to be suppressed if a prior event A has been detected. An A event should always occur two seconds before a B event.

Normally, it would be sufficient to remember that A had arrived, and when B arrives, discard it. But what if there has been no previous A event? Was no corresponding A event generated, so that the B event should be forwarded, or has the A event been delayed? If no A event has been received, it is necessary to hold B until there is no possibility of an A event arriving. If an A event doesn't arrive within some configured time, the B event should be transmitted. The permissible delay after which the B event will be forwarded is dependent upon the probable worst-case delay for any particular management network.

For any event, the delay imposed by the management network is known as the *transit delay* for that event. All events are considered to have a transit delay. ECS processes delayed and incorrectly ordered events in real time. Events can be reordered within the engine and held by individual nodes until related events are delivered even if subject to significant transit delays. Since events from multiple sources can be correlated, it is necessary that these sources have their clocks synchronized within known limits.

Information Concentration

ECS allows all pertinent information to be concentrated during correlation, with redundant and superfluous information suppressed. The useful information can be forwarded to management systems using either new events created by ECS or original events modified by ECS.

In the above scenario, a B event was to be suppressed if a corresponding, previously generated A event had been received within a specified time window. It is possible that the B event contains some useful information not contained in the A event. ECS allows the correlation engine to create a C event that contains the useful information from both the A event and the B event, along with any public information. In this situation, both the A and the B events would be suppressed, while the C event is transmitted. The event volume is reduced without losing any management information. Alternatively, the information content of event A could be modified to include the useful information from event B.

Correlation Circuit

A correlation circuit is a set of interconnected and appropriately configured nodes. Fig. 3 shows a typical correlation circuit. Events enter a circuit (visually) on the left through source nodes, flow through a user-specified interconnection of nodes, and depart the circuit through sink nodes.

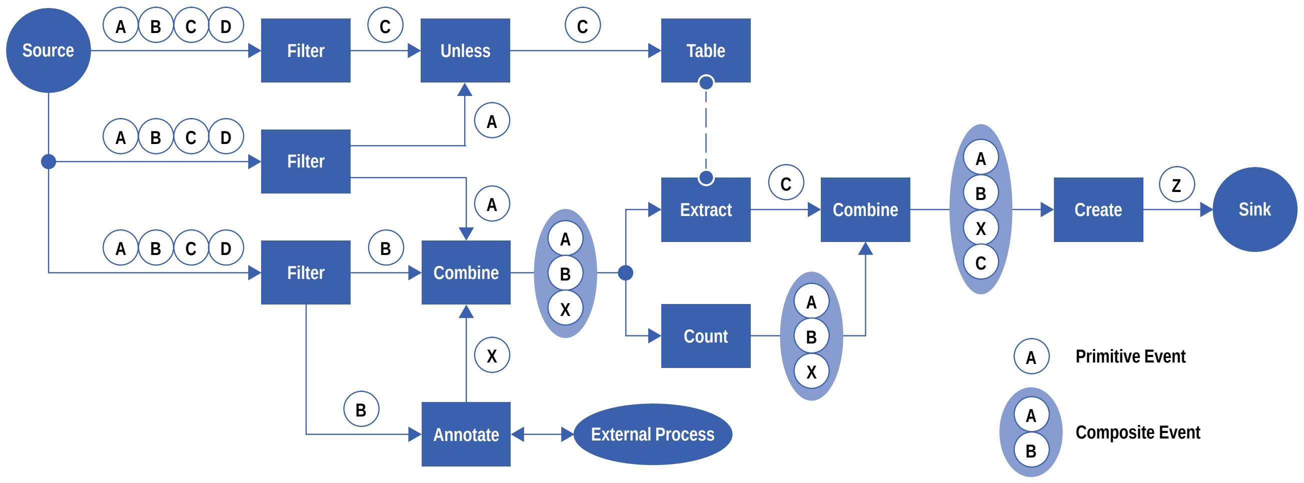


Fig. 3. *Typical correlation circuit.*

Events can be distributed down multiple paths within a circuit, and separate paths can be collected together. The event manipulation is controlled by the nodes in any given path. Events can be discarded, combined, created, and so on based upon the node types, the configuration of the nodes, and the dynamic conditions within the correlation circuit.

The correlation circuit paradigm results in a very intuitive implementation using a visual design and debugging interface that allows the circuit designer to see what operations depend upon each other.

Correlation Nodes

There are fifteen primitive node types, each type having a unique logical function (see sidebar: ***Correlation Node Types)***. For example, a filter node will either forward or suppress an event depending upon its configuration. The primitive nodes can be combined to create powerful and efficient correlation circuits, in a manner similar to RISC processors, in which simple instructions are used to deliver powerful solutions.

Circuits begin and end with *source* and *sink* nodes, respectively. Events can be filtered conditionally or explicitly with *unless* nodes and *filter* nodes. Event ordering can be modified with a *delay* node, and events can be stored for future reference with the *table* node. Events stored in a table node can be extracted later with the *extrac*t node. Where information external to the correlation engine is required, the *annotate* node is used. Since ECS is used to assemble and consolidate information from multiple events, the *combine*, *rearrange*, *modify*, and *create* nodes are essential. Decisions are often based on the values maintained by *count* and *rate* nodes. Testing for the absence of expected events is facilitated with the *clock* node.

Event filtering is a degenerate case of event correlation requiring only filter nodes (and source and sink nodes) to complete the circuit. Filtering only considers discrete events.

Compound Nodes

A compound node encapsulates a subset of a correlation circuit and defines a new node type with user-defined functionality (see Fig. 4). In the ECS Designer, a compound node can be exploded to show the contained circuit, which may contain additional compound nodes. Compound nodes can be nested to any number of levels.

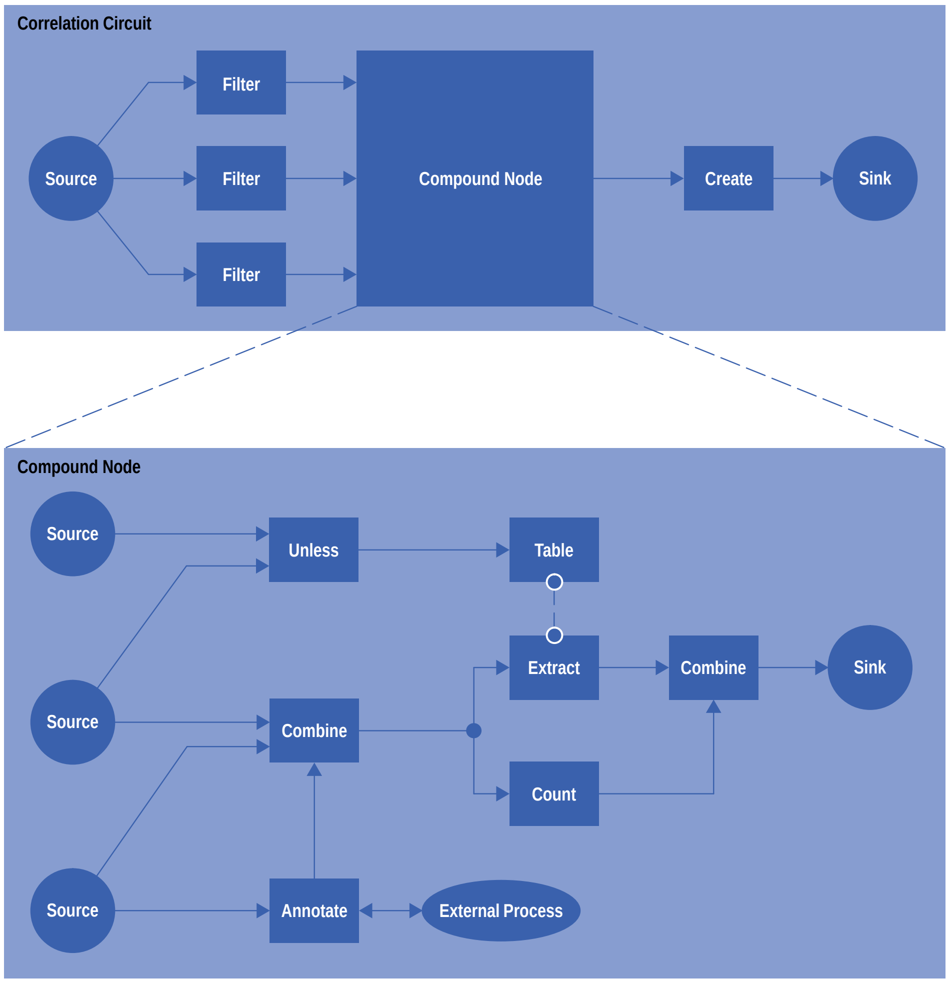


Fig. 4. *A compound node contains a correlation circuit that defines a new node type with user-defined functionality.*

An empty compound node can be placed on the ECS Designer canvas and interconnected with other nodes. The compound node can be exploded to add the required internal circuit. Alternatively, an interconnected set of existing nodes can be selected and converted to a compound node. This supports both top-down and bottom-up design methodologies, information hiding, abstraction, modular design and construction, improved circuit readability, functional reusability, and component testing.

Architecturally there is no difference between a top-level correlation circuit and a compound node. A circuit can be considered to be a compound node and vice versa. The contents of the compound node consist of an interconnected set of nodes including source and sink nodes. The source and sink nodes are connected to ports on the outside of the compound node. The ports are used to connect the compound node into a higher-level circuit. At the top level, the ports are connected to the external environment (see Fig. 5).

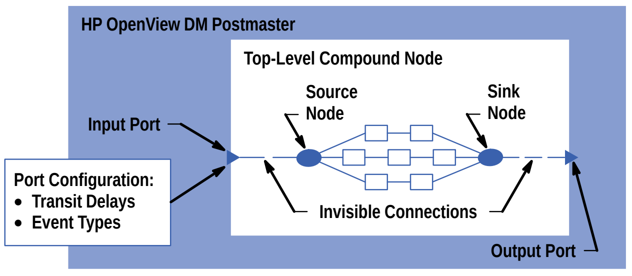
A compound node can be added to a library of compound nodes. A library can be selected and added to the tool bar of the ECS Designer so that the member nodes can be selected and used just like any of the primitive nodes. A library compound node can be copied, allowing local customization, or used by reference to the library copy. Reference use allows single-point maintenance of the library copy.

Node Configuration

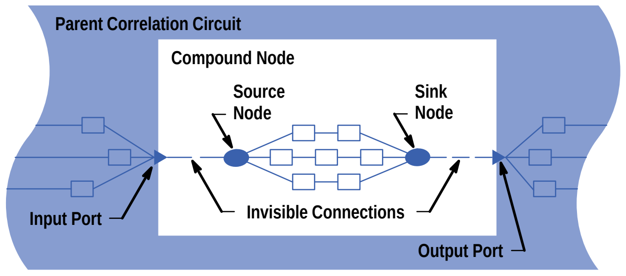
Each instance of a node must be configured to the user's requirements. Node parameters are configured by the user to customize the functionality of the instance. Each node has a set of ports, which can be connected to ports of other nodes, with connections to some ports being required before the circuit will compile. The components of the count, unless, and table nodes are described in the three sidebars: ***Count Node***, ***Unless Node***, and ***Table Node***.

Parameters

All nodes are customized to specific requirements by setting the values of node parameters via the node's configuration dialog. Parameters are either evaluated by the engine at run time, or repeatedly whenever an event arrives at the input port of the node. Thus, parameters are either *static* or *dynamic*.



(a) Top-Level Circuit Port Connections



(b) Compound Node Port Connections

Fig. 5. *Compound node port connections. (a) Top-level. (b) Lower-level.*

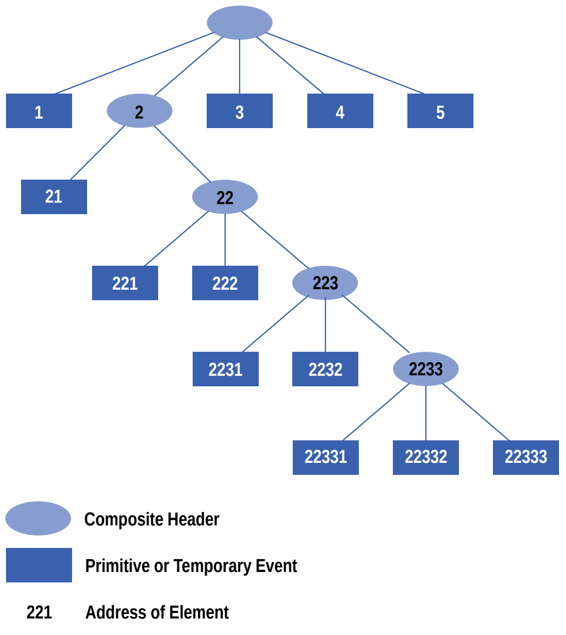


Fig. 6. *Structure of a composite event.*

Static parameters typically set resource and operational limits, such as the size of a table node, the frequency with which a clock node generates an event, how long an unless node should wait for an inhibiting event, and so on. Dynamic parameters define the operational behavior of a node. For example, a dynamic parameter of a filter node is a Boolean condition that is evaluated when an event enters the input port. The expression can take the incoming event as an argument, along with any other data from throughout the circuit. The result determines whether the event is forwarded to the true output or the false output.

All parameter values are actually expressions that can use references to values stored in the *data store* and to relationships stored in the *fact store* (see sidebar: ***Fact Store and Data Store***). For example, where a value should be supplied for a parameter, the name of a variable defined in the data store can be used. The data store entries can be changed without changing the configuration of the node. Dynamic node parameters are evaluated whenever a new event arrives at the input port of the node. If these parameter expressions reference data store or fact store entries, updates to these stores will affect subsequent parameter evaluations. This allows the behavior of a correlation circuit to be modified dynamically.

Like any primitive node, a compound node can have parameters that must be set or configured when the node is instantiated. The parameters are defined and documented by the designer of the compound node.

Ports

All nodes have one or more ports, which are visible in the ECS Designer. Nodes are interconnected via ports appropriate to the required functionality. Depending upon type, nodes can have input ports, output ports, error ports, reset ports, and other types of ports. Normal operations occur through the normal input and output ports. For a filter node, the configured expression should evaluate true or false, causing the incoming event to be routed to the true output or the false output port as appropriate.

Where run-time errors occur in evaluating the expressions configured for particular node instances (not everything can be known before run time), an event will be output through an error output port.

Except for the combine and compound nodes, primitive nodes have a fixed number of ports. The combine node can have up to 50 input ports (combining event streams). The compound node can have up to 50 input and output ports to support the encapsulated functionality.

Reset Ports

Delay, unless, combine, and annotate nodes can hold events in memory associated with an input port pending some condition becoming true. Table nodes can hold events in long-term memory.

When the engine is to be stopped or the correlation circuit is to be modified, the future conditions can now never be true, and the applicability of stored events is indeterminate in the context of the modified circuit. Even if the circuit were to be stopped and restarted without change, the potential for pertinent events to have been missed invalidates the state of the correlation. Critical events that should have been output may now be discarded.

It is necessary to be able to output the stored events before engine reset or reload so that potentially critical events are not lost. All nodes that store events have reset input and reset output ports. If an event is forwarded to a node's reset input port, any stored events will be output or discarded in a defined manner. The reset event will be output via the node's reset output port, possibly to a downstream node's reset input port, allowing a reset circuit to be added to an operational circuit.

Public Data

The nodes of a correlation circuit typically make decisions and perform actions based upon the arriving events, the information within the events, and the current time of the engine. Other data is also available to the dynamic node expression parameters to control node processing, including node attributes, the data and fact stores, and annotation data.

Node Attributes. A node can export one or more data values for use by expression and condition parameters configured for other nodes throughout the circuit. The count node exports a count attribute which increments or decrements for each arriving event. The table node exports two attributes: a count attribute whose value is the number of events currently stored and a contents attribute whose value is a list of all events stored in the table.

Compound nodes can export the attributes of any contained nodes as attributes of the compound node. If they are not exported, the attributes of internal nodes are private to the compound node and will not be visible outside the compound node.

Data and Fact Store. The data store contains entries of name-value pairs and the fact store contains entries of name-relation-name triples (see sidebar: ***Fact Store and Data Store***). These stores operate as in-memory databases and are accessible globally throughout the correlation circuit.

The data store entries allow values to be referenced by name, so that particular values need not be known when the circuit is designed. Using indirection through the data store also allows correlation circuits to be reused at multiple sites, with specialization via appropriate data store values.

The fact store allows the relationships between objects to be tested by node parameter conditions and expressions. Typically the entries will be used to reflect network topology information, and will allow event relationships to be determined dynamically without building the network model information into the actual correlation circuit.

Annotation. An annotate node (see sidebar: ***Annotation***) can be used to obtain data from outside the correlation engine for use within the engine. This data will be used to make correlation decisions, or to add to created events or modified events.

Event Types

Several event types can logically exist within a circuit. These include primitive events, composite events, and temporary events.

Primitive Events. A primitive event is a single event as it entered the correlation engine, possibly with data values modified within the circuit, or an event created within the engine, suitable to be returned to the external environment. ECS currently supports CMIP (Common Management Information Protocol, ISO/IEC 9596-1) and SNMPv1 (Simple Network Management Protocol, version 1) event types. The engine isolates the external event type from the internal functionality using specific decode and encode modules for each event type. This modular design allows additional event types to be supported in the future. Since the internal processing is not dependent upon the format of primitive events, it is possible to correlate events of any supported type with events of any other supported type.

Composite Events. A composite event is an ECS internal mechanism that allows multiple events to be collected into a single addressable structure in which all members are accessible (Fig. 6). The event aggregation capability is fundamentally important for ECS. Collecting and processing multiple events as a single event allows all important information to be collected together. Members of a composite event can be primitive, composite, or temporary events. A composite event is only defined within a correlation engine, and cannot be output from a top-level circuit back to the environment. Composite events can be passed into and out of compound nodes.

Temporary Events. Where an event is required as an internal container for data, or where a trigger event is required, the engine will create a temporary event. For example, the clock node will emit a temporary event at each clock period. There is no relevant data in this empty temporary event. It can be used to trigger correlation activity elsewhere in the circuit. Where results are returned in response to a request by an annotate node, a temporary event is created to hold the data and returned to the circuit as a component of a composite event. A temporary event can enter or leave a compound node, but it cannot be output from a top-level circuit back to the environment.

Enhancing Event Information

A fundamental value proposition of ECS is that event information can be enhanced. What this means in reality is that all available information—for example, all information relevant to some network fault condition—is consolidated from multiple time-separated events, the data store and fact store, and data external to the engine via annotation.

When all pertinent information has been assembled (and all superfluous data discarded), it must be forwarded to interested operations systems. This can be done by:

- Creating a new primitive event containing the consolidated information, using the create node to create the event and copy the data into the event.
- Modifying the data values in an existing primitive event, using the modify node to change the values of the event's attributes before the event is output.

The input primitive events, which each contain only a fraction of the total relevant information, can be suppressed. Only the new or modified events are forwarded to interested management entities. The result is that the events actually delivered to management systems contain enhanced information content.

To aggregate all pertinent data into the delivered events, it is necessary to be able to collect the information together and process it through the engine as a single data unit. This allows correlation decisions to be applied to the logical block as a single unit, providing major efficiencies in circuit design and processing loads. Composite events are used to aggregate events into a single unit.

Event Processing

When an event is received by the correlation engine it must be decoded from the specific format (BER encoded*) sufficiently to determine the event creation time and the event identification. These values are fundamental to the operation of ECS. The creation time must be known to allow the time relationships between events to be known. The event identification can be used explicitly to control which branch of the circuit the event will logically enter.

HP OpenView ECS has been designed for high performance. Event encoding and decoding, and event copying within the engine, are implemented using a just-in-time encode and decode mechanism and sophisticated systems of header structure lists, event lists, pointers, and reference counts. An event is not fully decoded if not required by the correlation rule parameter expressions. References to events pass from node to node, rather than active events or copies of events.

When an event is forwarded down multiple paths in a circuit, reference counts are incremented. Only when one of these logical copies is modified (say with the modify node), is the event duplicated before modification. When the reference count is decremented to zero, possibly when the event is output from the circuit, the event is removed from the event list.

Retained Events

The transit delay of an event is defined as the number of seconds between the creation time of an event and the time that the correlation engine receives the event, assuming that both clocks are synchronized.

The example previously used considered the case in which an event A has arrived and a consequential event B is suppressed when it subsequently arrives. The correlation must consider the permissible transit delay range for event A to cover the situation in which event A arrives after event B. This requires that either event A or event B be retained in the circuit at the point where the condition is being tested. In a real-time engine, in which memory resources must be conserved, the event

* BER stands for Basic Encoding Rules (ISO/IEC 8825, ITU-T X.209). The BER define how ASN.1 (Abstract Syntax Notation 1, ITU-T X.208) data types are encoded to be transported on the network. Both of the primitive event types supported by HP OpenView ECS, that is, SNMP traps and CMIP notifications, are encoded using BER.

should be retained only while there is a possibility that it may be required in an active correlation, and automatically destroyed when no longer required.

A circuit must be configured with a transit delay window, which acts as an initial filter to eliminate any events with creation times outside this window relative to the correlation engine time. The circuit transit delay window is propagated into the circuit to calculate the transit delay limits on all nodes whose operation depends on event time differences. If a node imposes an additional time window for event comparisons (e.g., an unless node allows an inhibiting event to occur at some time offset from the exciting event), the allowable transit delays for the subsequent circuit are automatically adjusted to include the additional possible transit delay.

Events can be retained in port or node memory pending some condition becoming true. Each such event will be examined at each engine clock cycle to ensure that the creation time relative to the engine time is within the computed transit delay window at that point. Events failing to meet this requirement are released from memory automatically.

Data Access and GDMO MIBs

The tests and comparisons performed by the nodes must allow events to be tested for content. ECS provides high-level access to any element of an event, and has language data types that map onto the ASN.1 data types in an event. In ECS, each addressable component of an event is referenced as a named attribute. For example, an event may be significant if its severity attribute has a value of critical. ECS provides a sophisticated mechanism that allows the designer to specify this test (for a filter node) as:

```
input_event(”severity”) = ”critical”
```

This Boolean expression extracts the severity attribute of the input event, tests the value of the attribute, and evaluates as either true or false.

The concept that an event has a series of attributes that have values that can be examined or modified is fundamental to ECS. The attributes of an event are all the components or elements of an event that are specified by the MIB (Management Information Base) that defines the event. The MIB is added to the underlying HP OpenView DM platform so that it can be accessed by the correlation engine. MIBs must conform to the GDMO model (Guidelines for the Definition of Managed Objects, ISO/IEC 10165-4, ITU-T X.722) or the HP OpenView DM platform will not accept them. Once part of the platform, ECS accesses the components of the events using the textual names from the MIB registration tree. The MIB is described in ***Article 6***.

Language

Underlying the ECS Designer GUI is a complex and sophisticated language called ECDL (Event Correlation Description Language), which supports the complete specification of the correlation circuit including all the dynamic node expressions and conditions. ECDL includes data types, operations, and functions that allow read access to all component data in the events as they traverse the circuit, and to all public data within the circuit. (Event attributes can be altered with the modify node.) The ECS Designer ensures that the circuit designer does not need to understand this language in great detail. The circuit is specified by the visual interconnection of selected nodes. Node parameters are specified wherever possible using simple ECDL constructs and supplied library functions written using ECDL or actually built into ECDL. Advanced users are able to create specialized reusable functions. The ECDL code produced by the ECS Designer is encrypted in source form and compiled for downloading to the correlation engine. Direct coding using ECDL is not supported and cannot be compiled.

Building and Testing Correlation Circuits

The ECS Designer (which includes circuit design and simulate modes) is a GUI that allows the circuit designer to use a highly productive intuitive paradigm to build a correlation circuit by interconnecting primitive and compound nodes (see Fig. 7).

In ECS Designer build mode, nodes are selected from the tool palette, placed on the canvas, and interconnected to form the correlation circuit. Subsets of the circuit can be encapsulated as compound nodes to improve readability, implement top-down design rules, or promote reuse. Each node must be configured with appropriate values or expressions for its parameters. The general flow of events is determined by the circuit layout, subject to the conditions imposed by individual node parameters.

When the circuit is complete, the ECS Designer can be switched to simulate mode. In this mode, events can be input to the circuit to allow visualization of the flow of events through the circuit.

Various visual techniques are used to provide circuit operation feedback to the circuit designer. Events can be input in various modes: stepped by event or by time, free-run at selectable speed until a breakpoint, and others. The status of each node can be examined at any time. For instance, the contents of table nodes can be examined, the number of events through various ports can be checked, and so on.

In the simulate mode, events are input to the circuit from an input event log. The circuit designer can view both the input events and the correlated output events by means of associated event browsers.

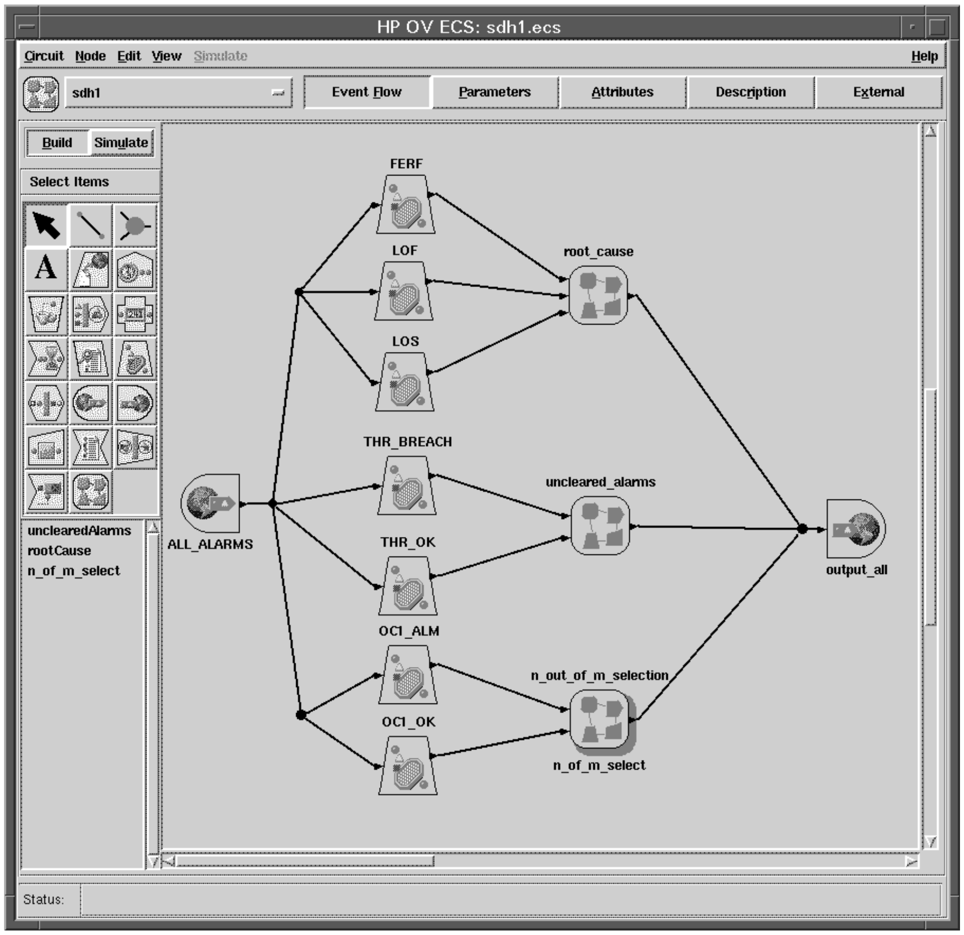


Fig. 7. *Constructing a correlation circuit using the ECS Designer GUI.*

The simulation is performed using a fully functional correlation engine, except that the engine does not free-run. The engine's notion of time is under the control of the simulator.

When a circuit has been developed and tested, the ECS Designer is used to compile the circuit so that it can be down-loaded into correlation engines, possibly in remote locations.

The event log that is input to the ECS simulator is in a structured ASCII format, allowing the events to be manually created or edited. It is desirable to use a log of real events, and to collect them automatically. Support is provided to collect real events in the required format. It may be necessary to make some simple edits to this event log to simulate the possible worst-case transit delays.

HP OpenView DM Interfacing

The correlation engine is integrated with the HP OpenView Distributed Management (DM) postmaster, ensuring that correlation is applied at a common point so that all events can be subjected to correlation (see Figs. 2 and 8). A correlation engine can be installed wherever an HP OpenView DM platform is installed. The distributed nature of HP OpenView DM event management services allows a distributed hierarchy of correlation engines to be readily implemented.

Adding correlation engines to the HP OpenView DM postmasters is transparent to existing agent and manager entities communicating via the postmasters, except that events can now be correlated. If the loaded correlation circuit were to pass all events, there would be no observable difference in the operation of these entities, or in the events being generated and received.

The HP OpenView DM platform is described in ***Article 1***.

Events entering the postmaster are routed to the correlation engine where they may be accepted into the engine depending upon the configuration of the circuit input ports (see Fig. 8). Confirmed CMIP events are immediately returned to the postmaster by the correlation engine. It is normally expected that a management entity will receive a confirmed event, and that the confirmation is returned as a consequence of some action having been taken, frequently by an operator. Typically, if the confirmation is not returned, the agent entity that generated the event will issue another (possibly different) event. The operation of the agent entity may be effected by the absence of the confirmation. If confirmed events were accepted into the correlation engine where they were subsequently suppressed as part of the correlation, the confirmation would not be

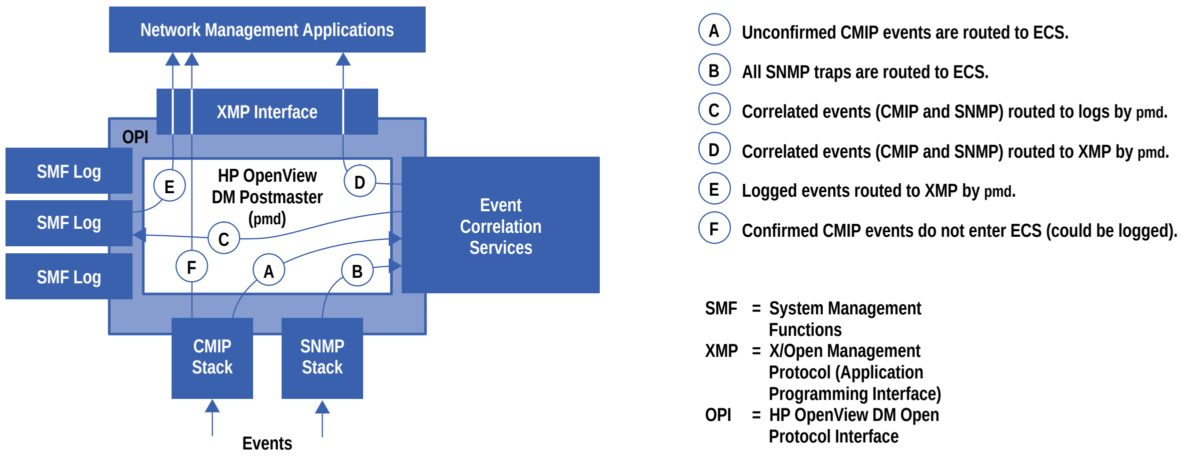

Network Management Applications
XMP Interface
OPI
SMF Log
SMF Log
SMF Log
HP OpenView DM Postmaster (pmd)
Event Correlation Services
CMIP Stack
SNMP Stack
Events
A Unconfirmed CMIP events are routed to ECS.
B All SNMP traps are routed to ECS.
C Correlated events (CMIP and SNMP) routed to logs by pmd.
D Correlated events (CMIP and SNMP) routed to XMP by pmd.
E Logged events routed to XMP by pmd.
F Confirmed CMIP events do not enter ECS (could be logged).
SMF = System Management Functions
XMP = X/Open Management Protocol (Application Programming Interface)
OPI = HP OpenView DM Open Protocol Interface

Fig. 8. *Event routing to and from HP OpenView Event Correlation Services.*

returned since the normal receiving entities would never see the event. Conversely, if the correlation engine generates the confirmation, the agent entity can modify its function based upon the assumption that an operator has seen the event and taken appropriate action.

The input ports of the correlation engine must be configured to accept all events received or they will be filtered out and discarded by the engine. Where significant events are expected for which correlation has not been designed into the circuit, a branch of the circuit can be arranged as a pass-through to transmit all events not explicitly handled by the other circuit branches.

Acknowledgments

I would like to thank Keith Harrison and Michele Campriani of HP Laboratories, Bristol for developing the theory upon which this technology is based, and for their encouragement and assistance in bringing the technology to market.

Reference

1. K.A.Harrison, *A Novel Approach to Event Correlation*, HPL-94-68, HP Laboratories, Bristol, England, July 1994.

▶ Go to First Sidebar
▶ Go to Article 5
▶ Go to Table of Contents
▶ Go to HP Journal Home Page