Fault Management System using Semantic Publish/Subscribe approach

by

Wei Tai, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in partial fulfillment

of requirements

for the Degree of

Master in Science in Computer Science

University of Dublin, Trinity College

September 2007

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

_____

Wei Tai

September 14, 2007

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Wei Tai

September 14, 2007

Acknowledgments

I would like to thank my dissertation supervisor, Dr. Declan O'Sullivan, for all his invaluable help and guidance over the period in which this dissertation was researched and written. I would like to thank Dr. John Keeney for his always willing to contribute his time and precious suggestions. I would like to thank my family for their continued support over the course of my school and college years. I would also like to thank my girl friend Ying for all her support and patient throughout the year. Finally, I would also like to thank my UbiCom classmates for help, support and for lightening the mood in serious times.

WEI TAI

University of Dublin, Trinity College

September 2007

Fault Management System using Semantic Publish/Subscribe approach

Wei Tai

University of Dublin, Trinity College, 2007

Supervisor: Declan O'Sullivan

With the rapid development of network technology and the ever growing demand on networks from both enterprise and network providers, current networks are increasing dramatically both in terms of scalability and complexity. However, traditional network management approaches (e.g. OSI approach) typically involve hierarchical manager/agent topologies and rely upon significant human analysis and intervention, both of which exhibit difficulties as scalability and complexity increases.

The drawbacks of traditional network management approaches limit their application to large scale networks. Currently many research groups, projects and academic institutes are applying themselves to develop new network management approaches which are high in scalability, performance, and intelligence.

This paper researches an alternative way to perform fault management in large scale networks using a semantic publish/subscribe system. A new distributed fault management system architecture based on a semantic publish/subscribe paradigm has been proposed; and, a new distributed event correlation scheme with guessing ability has also been proposed.

The evaluation of this project shows that the performance of the proposed fault management system increases dramatically with the increase of the number of fault management servers. We conclude that the proposed distributed fault management system holds promise in performing large scale network management. However, more research is still needed to be done in order to achieve a system suitable for supporting autonomic approaches, which are expected to be core to the next generation of management systems.

Contents

Acknowledgments ..... **IV**

Abstract ..... **V**

List of Figures ..... **IX**

List of Tables ..... **XI**

Chapter 1 Introduction **1**

Chapter 2 State of the Art **5**

2.1 Event Correlation Schemes 5
2.2 Network Management/Fault Management Architecture 10

Chapter 3 Project Background **13**

3.1 Background scenario and scenario modeling 13
3.2 Technology Background 16
3.2.1 Ontology 16
3.2.2 Publish/Subscribe system 19
3.3.3 Simple Network Management Protocol 20
3.3.4 BGP/MPLS VPN 22

Chapter 4 Distributed Fault Management System **24**

4.1 Physical Architecture 24
4.1.1 Fault Management Servers 25
4.1.2 Front End Servers 26
4.1.3 KBN 27
4.2 Software Architecture 27
4.2.1 Software Architecture of FMS 28
4.2.1.1 Event Normalizer 28
4.2.1.2 Knowledge Base 30
4.2.1.3 Service Organizer 32
4.2.1.4 Event Correlator 33
4.2.1.5 Trouble Shooter 34
4.2.1.6 Configuration Manager 35

4.2.1.7 Subscription Manager 36
4.3 Distributed Correlation Scheme 38
4.3.1 Scheme background 38
4.3.2 Scheme Overview 39
4.3.3 Technology Background and Terminology Definition 40
4.3.4 Correlation Task Analysis 46
4.3.5 Correlation Schemes 48
4.3.5.1 Direct Correlation Rule calculation 50
4.3.5.2 Correlation Table Initialization 51
4.3.5.3 Delay Correlation 54
4.3.5.4 Correlation Result Merging 55
4.3.5.5 Event Guessing 59
4.3.5.6 Post-correlation processing 60
4.3.6 Delay correlation mechanism and delay window analysis 61
4.4 Correlation Expert Knowledge self-updating mechanism 66

Chapter 5 Implementation 68
5.1 Overview 68
5.2 Simulator Implementation 69
5.3 Service Organizer Implementation 71
5.4 Knowledge Base Implementation 75
5.5 Event Normalizer Implementation 77
5.6 Implementation Issues 79

Chapter 6 Evaluation 81
6.1 Performance Benchmark 81
6.1.1 Benchmark Criteria, Methodology and Results 82
6.1.2 Statistics Analysis 84
6.2 Feature Comparison 87
6.2.1 Architectural Comparison 87
6.2.2 Correlation Scheme Comparison 89
6.3 Evaluation Conclusion 90

Chapter 7 Issues 91

Chapter 8 Conclusion and Future work 93
8.1 Achievements 93
8.2 Highlights 93
8.3 Future work 94

Bibliography ..... 错误！未定义书签。

Appendix A Design of Test Cases 102

Appendix B Format of trap simulator configuration file.....103

List of Figures

Figure 1-1 Cooperation between components **4**

Figure 3-1 Scenario network **14**
Figure 3-2 Events that can occur in the scenario network **15**
Figure 3-3 Example Event Correlation Graph **16**
Figure 3-4 An ontology example **17**
Figure 3-5 Example Publication for CBN **20**

Figure 4-1 Physical architecture **25**
Figure 4-2 FMS Architecture **28**
Figure 4-3 Design of Event Normalizer **30**
Figure 4-4 Design of Knowledge Base **31**
Figure 4-5 Common Denominator design of Service Organizer **32**
Figure 4-6 Design of Event Correlator Service and its dependent components **34**
Figure 4-7 Subscription manager controls the correlation task **37**
Figure 4-8 Subscriptions of correlator A and correlator B **37**
Figure 4-9 Overview of event correlation **40**
Figure 4-10 Example Event Correlation Graph **41**
Figure 4-11 Correlation Rule Set representation of Figure 4-10 ..... **43**
Figure 4-12 DCR calculation using merging rule and substitution rule **46**
Figure 4-13 Correlation Task Analysis **47**
Figure 4-14 Three correlators example **49**
Figure 4-15 Correlation Result Substitution algorithm **57**
Figure 4-16 Correlation Result merging process **57**
Figure 4-17 Event Arrival Patterns for event correlation **63**

Figure 4-18 FMN topology **65**

Figure 5-1 Implementation level architecture of this system **69**

Figure 5-2 Implementation level architecture of event simulator **70**

Figure 5-3 SNMP trap wrapped by KBN notification **70**

Figure 5-4 Implementation level architecture of event correlator **72**

Figure 5-5 Implementation Level architecture of Event Normalizer **79**

Figure 6-1 Average Branch Selection Time incremental model **85**

Figure B-1 Synthetic SNMP trap **103**

List of Tables

Table 4-1 Initial Correlation Table for Thread-1 ..... **51**

Table 4-2 Initial Correlation Table for Thread-2 ..... **54**

Table 4-3 Initial Correlation Table for Thread-1 when C is received after correlation window **55**

Table 4-4 Correlation Table in Correlation Result of C **56**

Table 4-5 Correlation Table in Correlation Result of D **56**

Table 4-6 Correlation Table for Thread-1 after Correlation Result Merging **59**

Table 4-7 Correlation Table for Thread-1 after Correlation Result Merging (Correlation Result of C lost) **59**

Table 4-8 Guessing Correlation Result for event C **60**

Table 4-9 Correlation Table for Tread-1 after Event Guessing process **60**

Table 6-1 Benchmark environment. **82**

Table 6-2 Test case execution result over 1 correlator **84**

Table 6-3 Test case execution result over multiple correlators **84**

Table 6-4 Performance Improvement **86**

Table A-1 Design of Test Cases **102**

Chapter 1

Introduction

Today the rapid emergence of novel network technologies and network services means that current networks get larger, more complex and more heterogeneous. Both the type and number of network elements needed to construct a network is increasing sharply. Not only traditional routers and switches but also dedicated network elements such as storage nodes are being used. In addition, the type and number of services provided by a network is also growing rapidly in order to satisfy different requirements. Traditional Web Access is not the only service that is being provided by provider network. The wide deployment of broadband access technology in the “last mile” and optical switch technologies greatly promote the emergency of new services such as Voice on IP (VoIP) and Virtual Private Networks (VPN).

In this context, the rapid development of the network both in complexity and scalability puts more rigid requirements on its Network Management System (NMS). In order to conduct management of the ever expanding network and maintain the services running on it, the network management system should have the ability to process thousands or even millions of network events per minute. Furthermore, it should hold more intelligence and thus relieve network operators from the numerous and heavy network management tasks. However, currently network management approaches such as OSI proposed manager/agent based network management model are mainly using centralized architectures. In centralized network management systems, most intelligence, such as event correlation function and fault recovery function resides in the centralized manager. This makes the centralized manager the

performance bottleneck of the whole system and limits both the throughput and event processing speed of the whole management system. Thus the centralized manager is no long suitable for the management of large scale network with large numbers of network services running upon it.

As the critical part of a Network Management System, the fault management system undertakes event correlation, and fault recovery operations. It is the major influence that affects the architecture of a Network Management System, and thus it is a major factor that affects the scalability and event processing speed of a network management system. Therefore, in order to enable a Network Management System more suitable for the management of current large scale networks, a more scalable and high-performance Fault Management System is urgently required.

Publish/subscribe systems have the potential to address the scalability issues that exists in traditional Fault Management Systems. By subscribing its interests to an underlying publish/subscribe system, an upper layer application will receive only the messages that satisfy its subscribed interests and gets rid of the disturbance of other irrelevant messages. In addition, the publish/subscribe system is an “addressless” transmission scheme, so no attention needs to be paid on the location of both publisher and subscriber, which enables the roaming and distribution of publisher and subscriber. All the aforementioned merits of publish/subscribe systems makes it a splendid candidate technology to enlarge the scalability of a new Fault Management System.

Thus, the research question posed for this dissertation was whether a semantic based publish subscribe system could provide the basis for a more scalable Fault Management System.

The Fault Management System proposed by this dissertation is novel in the architecture it is using and in the fault algorithm that has been developed. It is

constructed by combining an existing semantic publish/subscribe system [34, 40], a novel distributed correlation scheme developed in the project, and the standards based Simple Network Management Protocol (SNMP)[35, 36] that is traditionally used for the majority of network management.

SNMP is widely used by Network Management Systems to monitor state of network and perform configurations on network elements. Most current network elements provide support for SNMP. It keeps the network element states and configuration information in Management Information Bases (MIBs), and performs operations on information in MIBs through get/set protocol primitives. The SNMP conceals the heterogeneity in both hardware and software existing in different network elements and makes the network management transparent to the Network Management System. Therefore, it was selected to perform the management information transmission in the proposed fault management system.

The third part of the solution used in this dissertation is the novel and original event correlation scheme that has been designed to perform distributed event correlation. It distributes and coordinates the correlation task among multiple correlators that operate in parallel and reside on several different servers. By using the distributed correlation scheme, a correlation task will be split into several task snippets and have them running on several correlator concurrently. This will greatly increase both the correlation speed and throughput of a fault management system.

In the implementation, the Fault Management System is architecturally composed of three main parts: distributed event correlators, a Knowledge Based Network (KBN) and front end servers. The distributed event correlators on which the distributed correlation scheme will run are mainly in charge of the root cause reasoning. The KBN, as the underlying system, provides a semantic publish/subscribe service to upper layer applications. The front end servers will work as interpreters which are mainly in charge of the conversion between system specific messages such as SNMP

and KBN compatible notifications. Figure 1-1 illustrates the cooperation between the three parts.

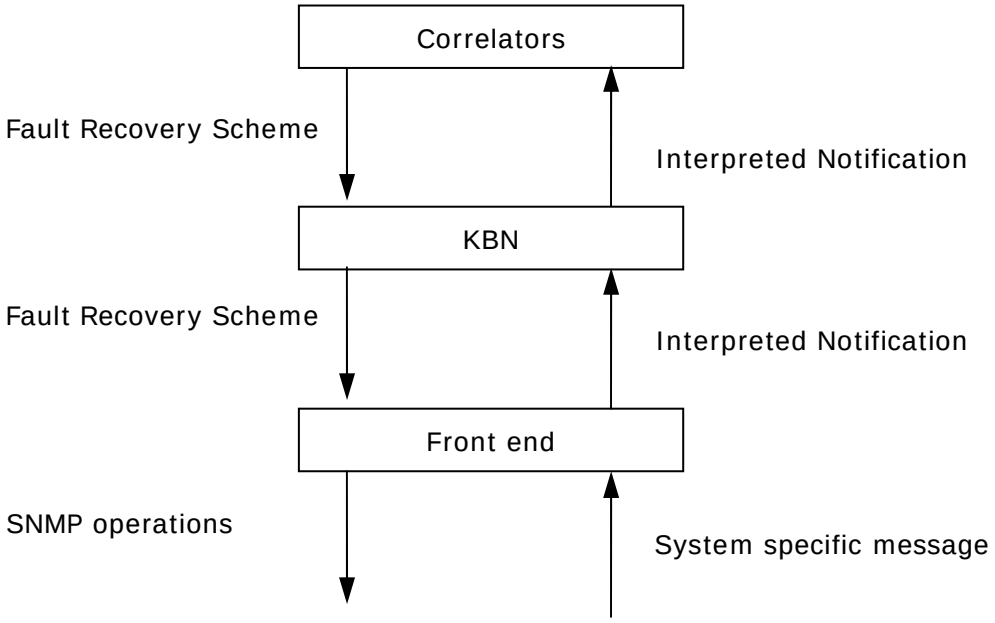


Figure 1-1 Cooperation between components

The rest of the dissertation is organized as follows. The state of the art of Fault Management Systems and Correlation Scheme will be illustrated in section 2. Section 3 includes some background information, describing the scenario that was studied to help understand the type of complexity of networks that exist, and background on the technologies used in the project. Section 4 describes the design of the proposed system, including the design of the fault management system and detailing the novel distributed correlation scheme that has been developed. Scenarios will also be used in this section to illustrate the cooperation of different parts in the proposed fault management system. The implementation is presented in section 5, followed by the evaluation to the proposed fault management system in section 6 describing performance measurements that have been undertaken and a feature comparison of the approach with the state of the art approaches. Section 7 outlines some of the remaining issues and section 8 presents overall conclusions and future work of this research.

Chapter 2

State of the Art

2.1 Event Correlation Schemes

Event correlation is the most important component in a Fault Management System. It condenses received events to a small set of more meaningful events. Furthermore, it can also identify the root cause from a set of received symptom events. Current event correlation schemes mainly can be categorized as: rule-based scheme, codebook approach, finite state machine approach, dependency graph approach, and Artificial Intelligence (AI) approaches.

- Rule-based scheme

Rule-based scheme is a traditional but practical event correlation scheme that uses a set of rules to match the events when they arrive at a correlation engine. The rule-based approach is a sophisticated technology but easy to understand. The rule has strong semantics and can express both causal logic and temporal logic among received events. However, it is low in scalability and sensitive to noise. In addition, the correlation rule is difficult to construct because the expert knowledge needed to create the correlation rules is extensive and hard to gather. Currently many commercial fault management systems are constructed based on rule-based scheme or rule-based expert systems, such as Event Correlation Service (ECS) in HP OpenView [10, 11], SDH network management system [10, 12], Sinergia Expert System [10, 13] and so on.

- Codebook approach

Codebook approach proposed by S. A. Yemini et al in [6] group all alarms caused by a fault into a complete alarm set. Each fault then is assigned a unique “code” which is represented through a binary vector. The events caused by a problem will then be treated as a “code” that identifies the problem. The correlation process is then turned into a “decode” process through determining which “code” has the minimum hamming distance with the incoming event set. The use of Boolean operation on symptom code makes the correlation process very fast. Besides, it uses minimum hamming distance to perform event correlation, thus it is more robust in the face of circumstances where events are lost or noise exists in incoming events. However, this approach is low in flexibility and scalability. Once the symptom for a fault is changed, all the codebook needs to be recompiled. Besides, the codebook approach can not correlate temporal events.

C. C. Lo et al in [7] improved the traditional codebook approach by adding in Event Causal Graph. The Event Casual Graph works as the knowledge base and from which the code for each fault will be generated. This approach alleviated the drawback that massive expert knowledge is needed to generate the code for a fault, and change the maintenance of codebook into the maintenance of Event Casual Graph.

- Probabilistic Finite State Machine

The event correlation approach proposed by I. Rouvellou and G. W. Hart in [1] models each fault using a Probabilistic Finite State Machine (PFSM). It has the ability to handle noisy event sequences, and uses probabilistic theory to select the right PFSM for the incoming fault sequence. The PFSM building process for each fault is a self-learning process using probabilistic theory, and the association information of fault to PFSM can be gathered from other systems or network experts input. This algorithm does not assume any knowledge of network structure and can automatically recognize the time pattern of alarms associate with a given fault.

- Network Element Dependency Graph

Several event correlation schemes are constructed based on Network Element Dependency Graph [2, 3]. In these schemes, Network Element Dependency Graphs are used to model the functional dependency among the network elements in the object managed network.

Positive Information Algorithm proposed by A. T. Bouloutas, etc in [2] is an alarm correlation algorithm using network dependency graph. They divided the object network into undividable components (either hardware or software), and construct dependency graph over the undividable components. Alarms issued by object network will be explicitly associated with location information to indicate the component that has fault or malfunction. Correlation will then be performed over the Network Element Dependency Graph using the information associated with each received alarm. Jaesung Choi, et al. in [3] proposed an alarm correlation algorithm that based on Positive Information Algorithm (PIA) given by [2], and they expanded the PLA algorithm with a new alarm candidate set selection algorithm using alarm casualty graph based on OSI managed object.

This approach can easy locate the fault and then use network element dependency graph to perform correlation. However, it needs the explicitly carry of location information in error messages, which currently in use network management protocols such as SNMP do not support for and thus limit the wide use of this approach.

- Deterministic Event Causal Graph/ Event Dependency Graph

Some Event Correlation approaches [4, 8] use Event Causal Graph to perform correlation. These approaches use Event Causal Graph to model the causal relationships among events, and then to correlate received events using the modeled causal relationships.

M. Hasan in [4] associated a rank with each node in causal graph. When a set of events is received, the correlation algorithm will try to deduct the causal relationship

using the event causal graph by iteratively select the node un-received node with all its immediate successors already existing in the set into the event set. The candidate selection process will stop when all the nodes satisfying the conditions been selected into the event set. The algorithm will then select the candidates with the highest rank as the root cause of the received events. This scheme is similar to codebook approach but it has the ability to correlate temporal events.

B. Gruschke in [8] proposed an event correlation graph using event dependency graph. Each node in the event dependency graph will be assigned to one of the only two states: correct or faulty. The correlation scheme is divided into two steps. First, the received symptom event will be mapped to the corresponding node in the event dependency graph, and change the state of the node into “faulty”. Second, a search algorithm will be started form the original received events and search for the node by which all original received symptom event will depend on. The result of the search algorithm will then be regarded as the output of event correlation. This approach is easy to understand but it requires too much expert knowledge to create the event dependency graph, and also there are circumstances that no common dependent events exist for the incoming events. Besides, this approach provides no support for the correlation of temporal events.

- Artificial Intelligence (AI) approaches

With the development of AI technology, it is also used in network management to perform the management for large scale network with complex management operations that can not be conducted by human-beings.

J. F. Huard in [47] proposed an approach that is based on XUNET and introduces the probabilistic AI approaches such as belief network. They constructed a separated belief network for each fault that could happened in the managed network such as link down, no connection and so on for fault identification. Once the small belief networks are done, they will be combined into a global belief network. They thought the issues

exist in this approach including the binding between physical network elements to the belief network modeling, the further researching on the decision engine algorithm and the refining and validating of underlying XUNET belief network.

D. W. Guerer, et al in [9] proposed a hybrid AI method that introduces AI technologies such as Neural Network, Bayesian Network and Case Based Reasoning into fault management. They divided the whole fault management flow into 7 steps: (1) alarm collection; (2) maintaining customer satisfaction via intermediate action; (3) alarm filtering and correlation; (4) fault diagnosis through analysis and testing; (5) generating fault recovery plan, and carry it out; (6) afterward fault elimination test; (7) record data and determine the effectiveness of current fault management function. For step 3, 4 and 5, one or more AI technologies will be assigned to perform operations contained in those steps. Through assigning probabilistic or symbolic AI technologies separately to different steps in fault management, this approach has more flexibility and capability in fault management than traditional deterministic approach.

Although holding promising, the current development of AI technology especially probabilistic AI technology is still not sophisticated enough for commercial use and more researches are needed to be put in.

- Hybrid approach

Some hybrid approaches have also been proposed to unify multiple approaches together so that complimentary approaches can bring new benefits when combined.

M. Yu, etc. in [5] proposed a hybrid event correlation by combining rule-based reasoning and the codebook approach. The Rule-based reasoning is used in low level correlation and is usually used to correlate events that occur within a network device, protocol and managed by an event management system or the events among different protocol layer but that have simple relationships. The codebook approach, however, is used for correlating events from different networks. It is usually used for high level

correlation. This approach adopts different correlation for different correlation level and scales well. However, it is still possible that there is too much workload in a single correlator.

2.2 Network Management/Fault Management Architecture

Currently it is rare to find single dedicated Fault Management Systems, but rather fault management is usually implemented as a core component in a Network Management System. Therefore, in this section, we will explore the state of the art of the architecture of Network Management Systems.

- Centralized

Centralized Fault Management is currently the most sophisticated and popular architecture for fault management. It has a single centralized server, and several agents spread in the managed network. The centralized server, or manager, includes most of the intelligence of the Fault Management System and thus is the most important part of the whole system. The agents, however, will act on behalf of and respond to the central manager to carry out the management operations. Because most of the intelligence is kept in the central server, this architecture is easy to construct and easy to manage. This architecture is suitable for the management of small scale networks. However, when applying to large scale network, thousands or even millions of events could arrive at the correlation every minute, which is highly possible to flood the centralized server. Thus this architecture is not suitable for the management of large scale network and the centralized server will become the performance bottleneck of the whole system.

OSI management framework [14] is one of the most popular used and sophisticated network management frameworks. It is standardized by the International Organization for Standardization (ISO). It provides a framework using object-oriented technologies

for network management. The architecture it uses is centralized manager/agent architecture. The main task of the manager includes issuing directives and receiving notifications and the main task of agent include carrying out directives, sending responses and emitting events or alarms. Currently many commercial network management systems and research such as HP OpenView, and the project proposed in [15] are constructed based on this framework. Being standardized a long time ago, OSI management framework did not put too much concern on the scalability of the managed network. Therefore, its centralized manager/agent architecture is not enough for the management for large scale network with thousands or tens of thousands network element.

- Hierarchical

Network Management Systems constructed based on hierarchical architecture have several management servers performing management operation on given network level, e.g. different protocol stack layer, or given different network domains. Higher level servers will then perform management over the whole managed network. This is a distributed architecture and can perform well for the management of large networks.

Telecommunication Management Network (TMN) [16, 17] is a framework that is defined for telecommunication network management and inter-communication. This framework is defined by ITU-T M.3000 series. It is constructed based on OSI management network, however, it has extended the OSI manager/agent management approach into hierarchical architecture. The building block that is defined in TMN, called the Operations System (OS), could be the manager of one peer building block, and it can also be the agent of another peer building block. This extension greatly increases the scalability and flexibility of this management framework when performing fault management and can be applied to the management of large scale network.

- Hybrid

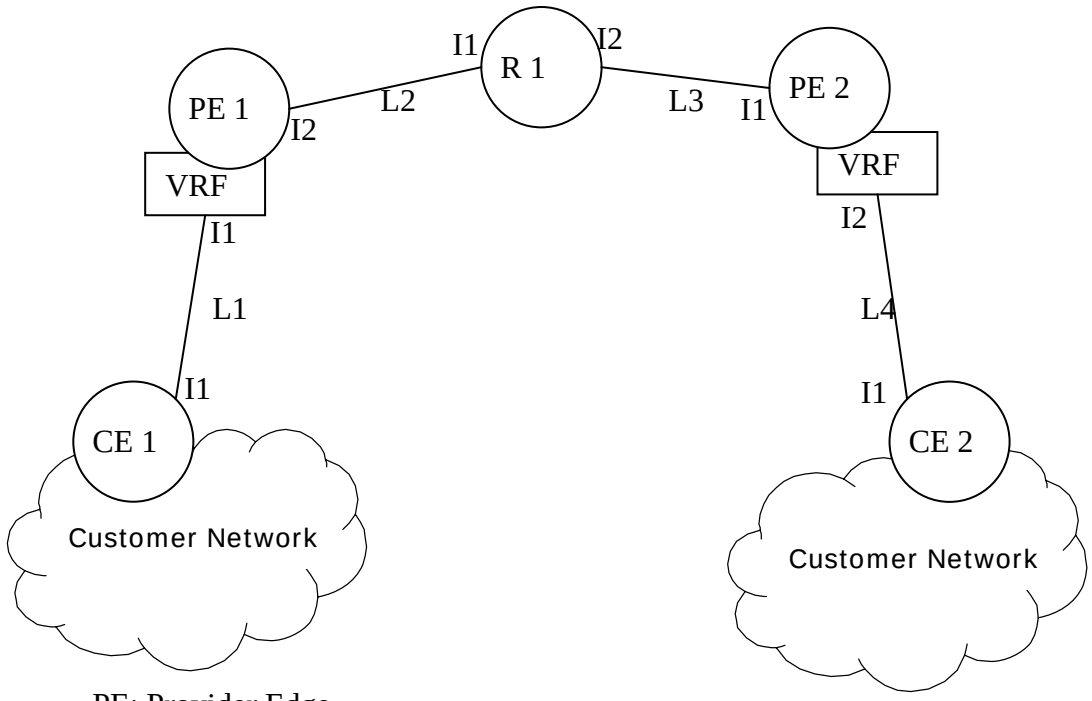
Some projects adopt the use of hybrid architecture. For example, the architecture used in the Madeira project [18], which is used for the management of wireless mesh network, combines peer-to-peer architecture with hierarchical architecture. In this project the management intelligence is distributed among the whole managed network into each network element. Peer network elements will be grouped into a cluster within which the peer-to-peer approach will be used. A cluster header will be selected from each cluster, which plays not only a peer role within the cluster, but also a head peer that communication with higher layer clusters. Clusters will be organized into a hierarchical architecture and there is a (or several) cluster in the highest management level which will be in charge of the management of the whole network. This approach can provide better scalability and robustness than using peer-to-peer structure or hierarchical structure alone, and besides it can improve the performance management operations such as fault correlation or fault recovery.

Chapter 3

Project Background

3.1 Background scenario and scenario modeling

In order to inform the research of the project a particular scenario was studied in order to understand the characteristics of the kind of complex networks that need to be managed. The scenario chosen was that of fault monitoring that would be needed for Border Gateway Protocol Multi-protocol Label Switching Virtual Private Network (BGP/MPLS VPN) [20, 43, 44, 45]. For verifying the feasibility of the designed solution and in order to have a target future real-world test, a simplified real world scenario (Figure 3-1) based on BGP/MPLS VPN was constructed.



PE: Provider Edge
CE: Customer Edge
I: Interface
L: Link
VRF: Virtual Routing and Forwarding table

Figure 3-1 Scenario network

As illustrated by Figure 3-1, this network has 5 routers in total. The PE1, R1 and PE2 are routers in provider backbone network and provide support for MPLS. The Customer network is connected to the provider backbone network using router CE1 and CE2. On the service level, two Label Switching Paths (LSPs) are constructed with LSP_1 as (PE1, R1, PE2) and LSP_2 as (PE2, R1, PE1). One VPN is set up to connect two geographically separated customer network into one network. We assume that this VPN is constructed using BGP/MPLS VPN and over LSP_1 and LSP_2.

Although this is a very simple network architecture, but there are massive number of diverse events that can be issued both the hardware and software level of this architecture. Figure 3-2 partially lists the events that can occur in the scenario network. These events, or notifications, will be illustrated using a format as EventName_SenderName.

Observable Alarms:	Hardware Level:	Software Level
1a. linkDown_If_PE1_1	1. If_PE1_1_down	1. Vrf_PE1_If1_misconf
1b. linkDown_If_PE1_2	2. If_PE1_2_down	2. Vrf_PE2_If2_misconf
1c. linkDown_If_PE2_1	3. If_R1_1_down	3. Bgp_cmpnt_PE1_down
1d. linkDown_If_PE2_2	4. If_R1_2_down	4. Bgp_cmpnt_PE2_down
1e. linkDown_If_R1_1	5. If_PE2_1_down	5. Ldp_cmpnt_PE1_down
1f. linkDown_If_R1_2	6. If_PE2_2_down	6. Ldp_cmpnt_R1_down
2a. mplsXCDown_PE1	7. Power_PE1_down	7. Ldp_cmpnt_PE2_down
2b. mplsXCDown_PE2	8. Power_PE2_down	8. Ilm_PE1_misconf
2c. mplsXCDown_R1	9. Power_R1_down	9. Ilm_R1_misconf
3a. mplsL3VpnVrfDown_If_PE1_1	10. Lk_CE1_PE1_disconnect	10. Ilm_PE2_misconf
3b. mplsL3VpnVrfDown_If_PE2_2	12. Lk_PE1_R1_disconnect	11. Mpls_fwding_cmpnt_PE1_down
4a. mplsLdpSessionDown_R1	13. LK_R1_PE2_disconnect	12. Mpls_fwding_cmpnt_R1_down
4b. mplsLdpSessionDown_PE1	14. Lk_PE2_CE2_disconnect	13. Mpls_fwding_cmpnt_PE2_down
4c. mplsLdpSessionDown_PE2		14. Vpn_cmpnt_PE1_down
5a. mplsTunnelDown_PE1		15. Vpn_cmpnt_PE2_down
5b. mplsTunnelDown_PE2		
5c. mplsTunnelDown_R1		
6. mplsBlackHoleDetected		
7. Lsp_PE1_PE2_down		
8. Lsp_PE2_PE1_down		
9. VPN1_disconnection		

Figure 3-2 Events that can occur in the scenario network

One fault occurring in the managed network will trigger multiple events. For example, link L2 down can cause the issue of event linkDown_If_PE1_2, linkDown_If_R1_1, mplsXCDown_PE1, mplsXCDown_R1, mplsTunnelDown_PE1, mplsTunnelDown_R1, VPN1_disconnection, LSP_PE1_PE2_down, LSP_PE2_PE1_down, etc. These events are all caused by the failure on link L2 and can be regarded as the symptom of link L2. In order to figure out this root cause from a set of symptom events, the relationship between symptom and root causes should be established to enable event correlation. In the proposed system, this relationship will be modeled using the Event Correlation Graph, which is constructed using an ontology and models all necessary fault information on the managed network into the fault management system. The Event Correlation Graph will be kept in the knowledge

base of the proposed fault management system. Figure 3-3 gives an example Event Correlation Graph which can figure out the root cause for the symptom event such as VPN1_disconnection, LspBroken_PE1_PE2 etc. For the sake of simplicity, only some of the events are used to construct the Event Correlation Graph in Figure 3-3. The graph needed for real-world use will be much larger and more complex than the one in Figure 3-3.

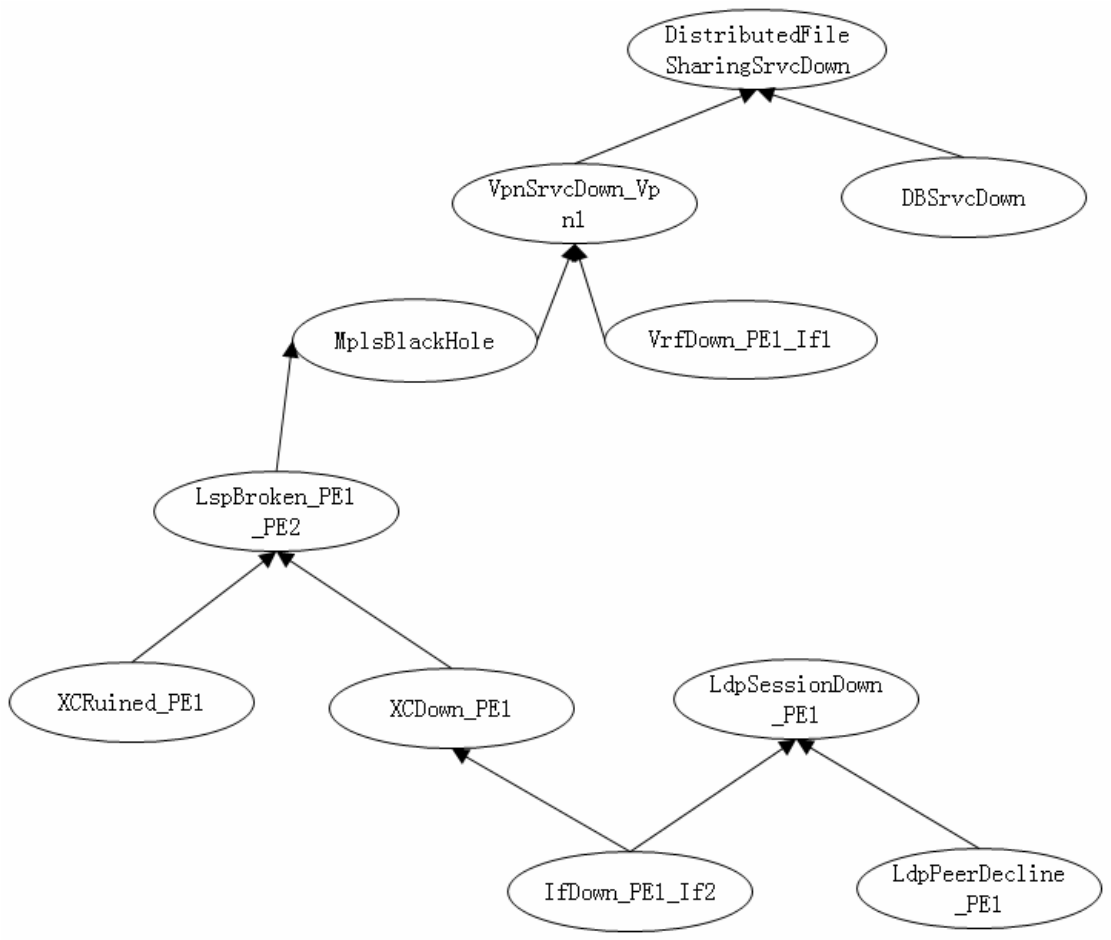


Figure 3-3 Example Event Correlation Graph

3.2 Technology Background

3.2.1 Ontology

Ontologies are about vocabularies and their meanings, with explicit, expressive, and well-defined semantics – possibly machine-interpretable [24]. According to the

definition given above, we can get that ontologies are a set of concepts as well as the relationship between them. These are usually implemented through classes, relations, properties attributes, and values, which are called resources. Figure 3-1 from [23] shows an ontology example.

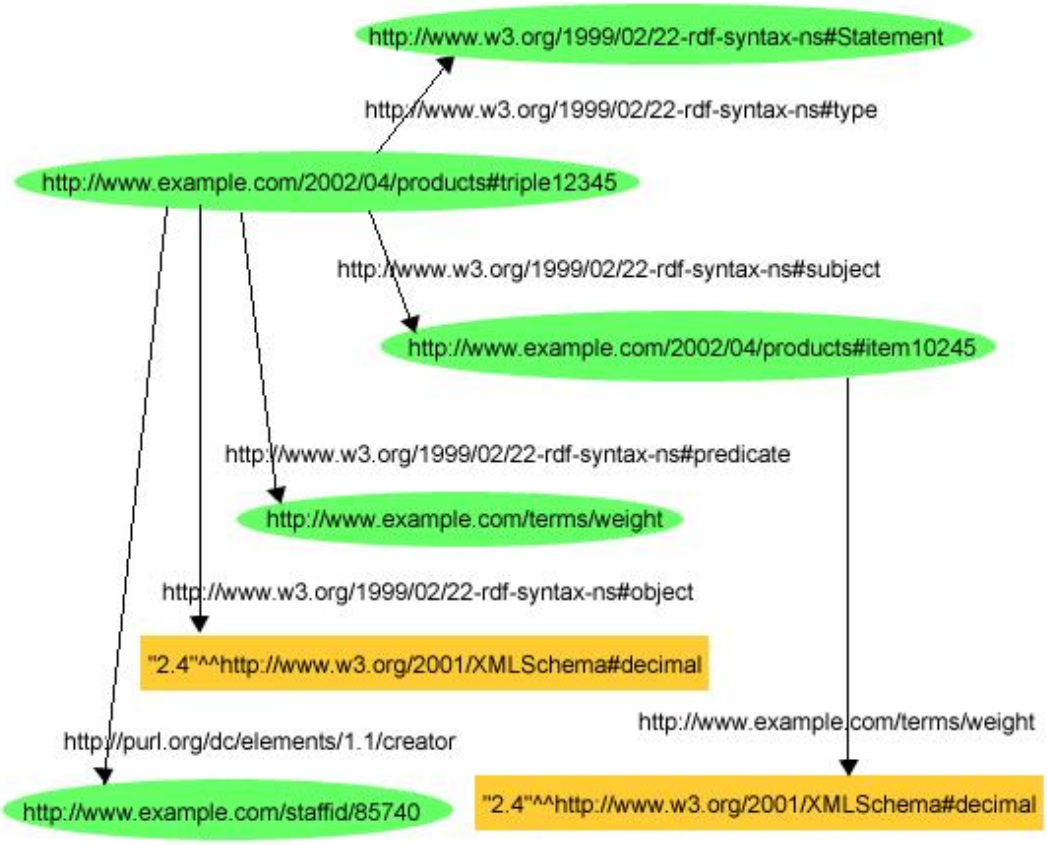


Figure 3-4 An ontology example

As illustrated by Figure 3-4, the ellipses represent objects, the arrows represent relations and the squares represent literal values. Usually, resources in ontology are represented by URIRef [25].

An ontology model can be represented by various formats. Three formats are the most common used: graph representation, triple statement representation and XML representation [27, 28]. Graph representation uses different shapes to represent different concepts and uses arrows which represent relations link them up. Figure 3-4 is an ontology that represented using graph. Triple statement representation uses a

statement which consists of a subject, a predicate, and an object to represent relationship between concepts. The statement representation is a verbal format of graph representation, and reads more like human language. The XML representation uses XML syntax for writing ontology. This representation is much more difficult for human to understand, but varieties of XML parsers enable the machine processing of ontology. Thus the mature standard and wide use makes XML an excellent media for ontology.

Many languages are developed to model ontology. They distinguish to each other not only in vocabularies, but also in the semantics representation ability. Resource Description Framework [23, 24, 26, 27, 28] is an simplest level ontology. It is developed to describe resources, for example, images or audio files and so on. RDF is weak at semantic representation for it only describe a concept and provide little support for the inference relationships between concepts, thus RDF Schema [23, 24, 29] is developed to as an extension to RDF. RDFS is language layered on top of RDF. It provides vocabularies such as rdfs:Class or rdfs:Property for defining concepts, besides, class concepts can relate to each other using subclass/superclass relationship, which enhanced the semantic representation ability of RDF. Web Ontology Language [22, 24, 28] has more vocabularies to express concepts and relationships between them than RDF and RDFS. OWL is constructed over RDFS, and some of vocabularies used in OWL already exist in RDF or RDFS. Otherwise terms are introduced by OWL. Vocabularies in OWL are further fined grained and are entitled with more semantics than RDF and RDFS. For example, OWL provides terms to express transitive relationship, inverse relationship, and symmetric relationship and so on, which all can not be expressed in RDFS. Currently, OWL has three versions – OWL Lite, OWL DL and OWL Full and OWL DL and OWL Full support for more terms than OWL Lite.

OWL is used to construct the routing ontology for underlying transmission middleware because of the requirements from underlying middleware. Besides OWL is also selected to construct the expert knowledge file for event correlation in the

evaluation implementation, which can also be done much easier through XML. However, OWL is still selected to model the correlation expert knowledge in the view of its reasoning ability could be used by expert knowledge base in the future work.

3.2.2 Publish/Subscribe system

Publish/Subscribe System [31, 32, 33, 34] is an appealing message transmission paradigm for it is both asynchrony and inherent loose coupling. A publish/subscribe system is usually composed of three components: publishers, subscribers and publish/subscribe middleware. Rather than sending messages explicitly to receivers by their address, publisher simply pushes messages into underlying middleware without the knowledge of subscribers' address (publish). The middleware will then classify messages into classes, and routing the classified messages to subscribers who paid interests in. In order to receive messages, subscribers do not need to keep senders' (publishers') addresses. It only needs to inform the underlying middleware the message classes it interests in (subscribe).

Subject-based publish/subscribe system is regarded as the earliest publish/subscribe system [32], which assigns predefined subjects to each message, and the underlying middleware routes messages according to the subject it belongs to. Content Based Network [31, 32, 33], however, assigns several tags to each event with each tag describing the information from specific aspect, and the underlying then can route the message according to the information carried in tags. Figure 3-5 is an example publication for CBN. CBN put more flexibility in the message routing, and currently many systems or technologies are developed or proposed after CBN, such as Elvin in [33] and Siena in [31] and so on. One thing worth mentioning that the underlying transmission middleware used in this project is an extension to Siena.

Semantic publish/subscribe system is a recently proposed. It extends CBN by the addition of ontology reasoning. Knowledge-Based Network (KBN) proposed in [34,

40] is a semantic publish/subscribe system that constructed based on CBN. It added ontology reasoning ability into CBN through the addition of ontology reasoning engine and thus events can be routed within KBN using “less specific”, “more specific” and “ontology equal” ontological operations. With ontology reasoning, KBN can be more flexible than CBN in event routing, and upper layer application can pay less attention to the addressing issue and thus increases the scalability of upper layer application.

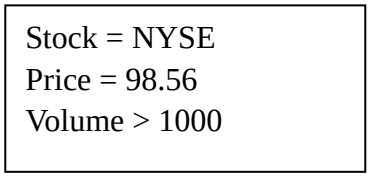


Figure 3-5 Example Publication for CBN

This project chooses KBN as its underlying message transmission system, which can decouple elements in managed network from fault management servers in the proposed fault management system, and thus increases the flexibility and scalability of the proposed fault management system.

3.3.3 Simple Network Management Protocol

Simple Network Management Protocol [35, 36] is an application layer protocol that is proposed by Internet Engineering Task Force (IETF), which is part of internet protocol suite. It is widely used by Network Management Systems to perform management on network elements. Currently three versions SNMP are in use, they are SNMPv1, SNMPv2 and SNMPv3.

SNMP perform network management using manager/agent structure, which is also the reason why OSI use the manager/agent structure. Manager will perform network management through issuing request to agents and receiving response from them. Agents are a small piece of software that resides on each manageable network

elements. It will receive requests from manager and then turn the received protocol PDU into the real operations on the management information. The agent addresses the heterogeneity among network elements, and makes the management transparent to upper layer Network Management System.

SNMP consists of three parts: an application layer protocol, a management information database schema, and managed data objects. The application layer protocol is mainly used by Network Management Systems to perform management operations [39] on management information that kept in managed elements. Currently, operation supported by SNMP includes Get Request, Set Request, Get Next Request, Get Bulk Request, Response Request, SNMP Trap, and Inform Request. The former 4 requests are originated from manager; whilst the last three operations are originate from agents. SNMP Trap (or Notification) is a mechanism that an agent proactively sends management information to the manager. It is usually used by agent to inform the occurrence of error or malfunction in managed network elements. The SNMP Trap will be considered as the main carrier of fault that occurred in managed network in this project.

The management information database schema in SNMP is termed the Structure of Management Information (SMI) [37]. It is a subset of Abstract Syntax Notation One which is a joint standard for describing data structure for encoding, representation, transmission and decoding data. SMI provides a set of rules for describing the structure of managed information independent to underlying network and machine specific encoding. The SMI is mainly divided into three parts: Module definition, Object definition and Notification definition. For more detailed description on SMI please refer to [37].

Management Information is stored as managed objects in the management information database which is termed as Management Information Bases (MIBs). It defines the managed objects that describe the behaviour and configuration

information of a given entity such as a SNMP protocol entity [41], a TCP entity [42], or an IP entity and so on. IETF gives some standard MIBs definitions in Request for Comments (RFC) documents. Besides, companies will also provide their own MIBs for the management of their products.

Besides SNMP, other management standards such as Distributed Management Task Force (DMTF) Common Information Model (CIM) are developed for network management. However, the simplicity has won SNMP a wide support from Network Elements such as Routers, Switches and so on, and thus a great number of network management systems perform network management using SNMP. For its simplicity and wide support, SNMP instead of other management approach such as CIM is selected to carry the management information and perform management operations.

3.3.4 BGP/MPLS VPN

Multiprotocol Label Switching [43, 45] is a packet switching technology. In MPLS forwarding paradigm, every packet will be forwarded in the MPLS network using the "Forwarding Equivalence Classes" assigned to it.

Unlike conventional IP forwarding, where the assignment of FEC is by performing the "longest matching" for the destination address of each arrived packet separately at each IP router, each packet will only be analyzed once at the ingress router of the MPLS network, and then be assigned a header with a label which represents the FEC. Packages will then be forwarded in the MPLS network by switching old label with new label without further analysis. At the egress router MPLS header will be removed form the package and the original package will be then forwarded to outside network, e.g. customer network. Because all forwarding in MPLS network is driven by label, this makes the forward speed much faster than traditional conventional network layer forwarding [43].

BGP/MPLS VPN [20, 44] is a technology that service provider uses MPLS based IP backbone to provide Virtual Private Networks (VPNs) services to customers. The VPN routes will only be kept in a data structure termed as Virtual Routing and Forwarding (VRF) in the router that reside at the edge of MPLS, and VPN routes will be propagated in the MPLS using Boarder Gateway Protocol (BGP) [46]. By doing this, the VPN routes knowledge will be transparent to the inner MPLS router. After VPN constructed, customer networks located at different geographical location will then be regarded in a single private network and access to each other.

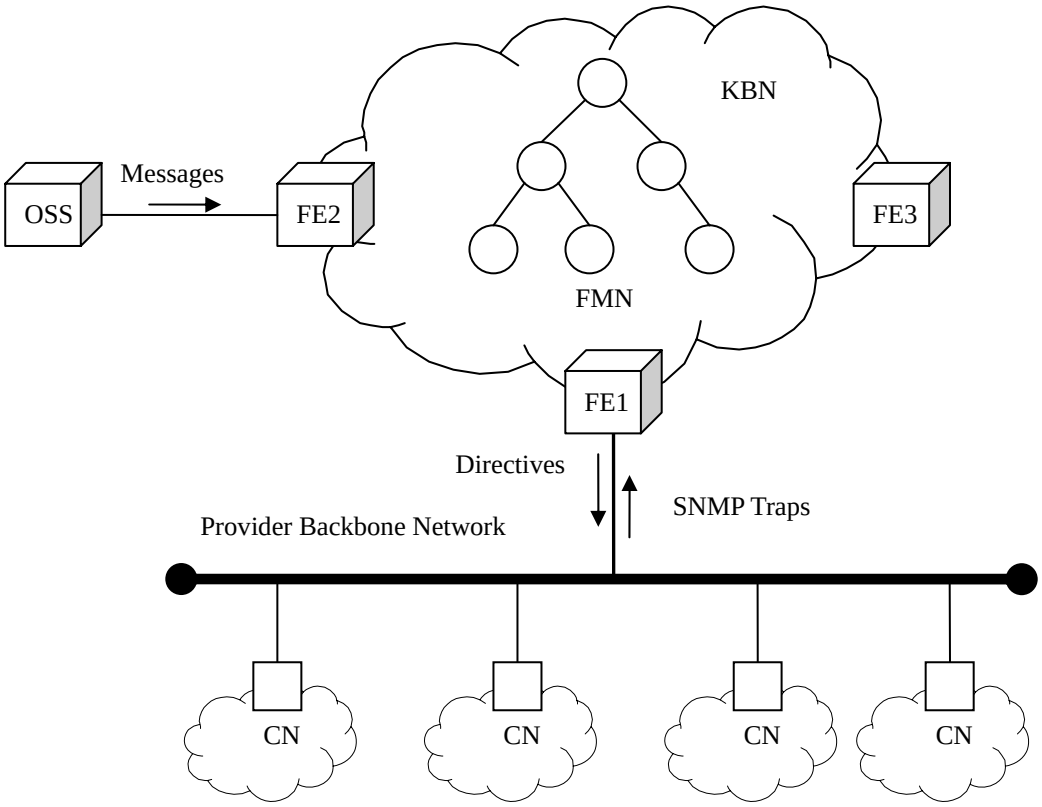
This project is constructed aiming at perform fault management over MPLS VPN and the real-world scenario is given in section 3.1.

Chapter 4

Distributed Fault Management System

4.1 Physical Architecture

Physically, the whole system (Figure 4-1) is designed to be composed of three main types of components: fault management servers, Knowledge Based Network (KBN), and front end servers. They will work together to provide the fault management services for the managed network.



FMN: Fault Management Network
CN: Customer Network
FE: Frond End

Figure 4-1 Physical architecture

4.1.1 Fault Management Servers

Fault Management Server (FMS) is the dedicated server that works as a service container. Services such as event correlation service, fault recovery service, logging service will run on it. A FMS can have multiple roles for example correlator, fault recovery policy holder, etc. The role for a FMS is determined by the context: when we talk about fault correlation, a correlator means a FMS, and so it is the same with other roles.

The managed network will be modeled from different aspects for different services in FMS. For example, all its events will be modeled as Event Correlation Graph for

event correlation use; in addition, all the directives that are supported by its elements will be modeled as fault recovery policy base for fault recovery use, and so on. All this expert knowledge will be kept into knowledge base that provides all required expert knowledge for services running on an FMS.

Multiple FMSs can form a Fault Management Network (FMN). The FMN is organized using hierarchical structure and each FMS within the FMN will assume the responsibility of providing the fault management at a given level and providing "processed" fault information, e.g. correlation result, for a higher level FMS. High level FMS will collect the "processed" fault information from low level FMSs and perform a correlation at a broader view. When distributing the fault management task within FMN, instead of being kept in every single FMS, the knowledge base will also be spread over the whole FMN with each FMS keeping the part it requires.

4.1.2 Front End Servers

Because messages from outside network can not be directed forwarded in KBN, there should be a dedicate server to translate messages from outside to KBN notification, in order to enable the forwarding of these messages. The Front End (FE) takes this responsibility that provides the mapping service between messages from outside systems, e.g. managed network, Operation Support Systems (OSSs), etc, and KBN compatible notifications. It works as a translator between outsider and the proposed fault management system. Instead of understanding the incoming messages, the FE only needs to perform the mapping between headers of incoming message and KBN notification tags. The mapping relationships are encoded using a policy approach so that they are replaceable and are easy to deploy.

Tags in KBN header varies with the type of messages. However, some common tags will exist in all KBN notifications. For example, in the current design "Message_Type", "Sender_Id" and "Payload" are three common tags.

The mapping process is a process of wrapping incoming messages by KBN headers. The mapping will not affect the content carried in original message, and the original message itself will also be carried in KBN notification as the value of a common tag “payload”. Tags in KBN header are only used for forwarding use in KBN. Once the KBN notification has been forwarded to the destination FMS, the KBN header will then be striped in Event Normalizer which we will mention to in section 4.2.1.1 and the original message will be delivered to upper layer services in FMS.

4.1.3 KBN

The KBN works as the underlying system and provides a semantic publish/subscribe message transmission mechanism for the inter-connection of FMSs and FEs. Every functional node running on it, e.g. FMS and FE, will work as a publisher or a subscriber or both a publisher and a subscriber.

An ontology that is constructed based on Event Correlation Graph will be used as the routing ontology for KBN. It will reside on every KBN node, and ontology comparison will be performed between the element in the routing ontology and the incoming message.

FMSs connect to each other using KBN and thus form a FMN. In order to get only their interested fault messages and “processed” fault information, the FMS will put a corresponding subscription to KBN so that these messages can be propagated to them, so it is the same with FE. We will give a detailed introduction to this later on in the next section.

4.2 Software Architecture

This section will give a full description on the software level architecture of the

proposed fault management system. Most software design of the proposed fault management system resides in the FMS and FE. Currently, only the design of FMS has been accomplished, so in this section, only the architecture of FMS will be given.

4.2.1 Software Architecture of FMS

As shown in Figure 4-2, an FMS is composed of 4 main components: Service Organizer, Configuration Manager, Knowledge Base and Event Normalizer. The Event Normalizer will decode the incoming messages and dispatch the incoming message to the right model; The Knowledge base stores all the expert knowledge on the managed network and fault management; The Service Organizer provides a running environment for fault management services; The Configuration Manager will be in charge of configuration operations. In the following text, we will give a detailed description of these 4 components.

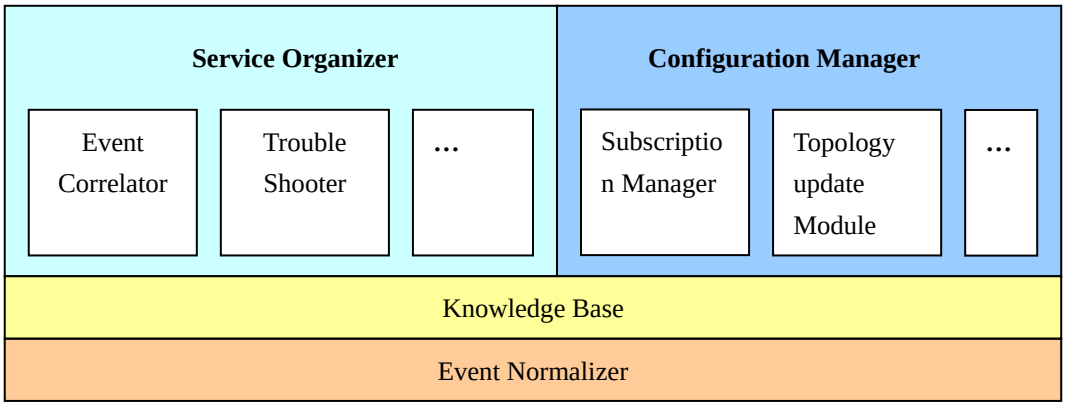


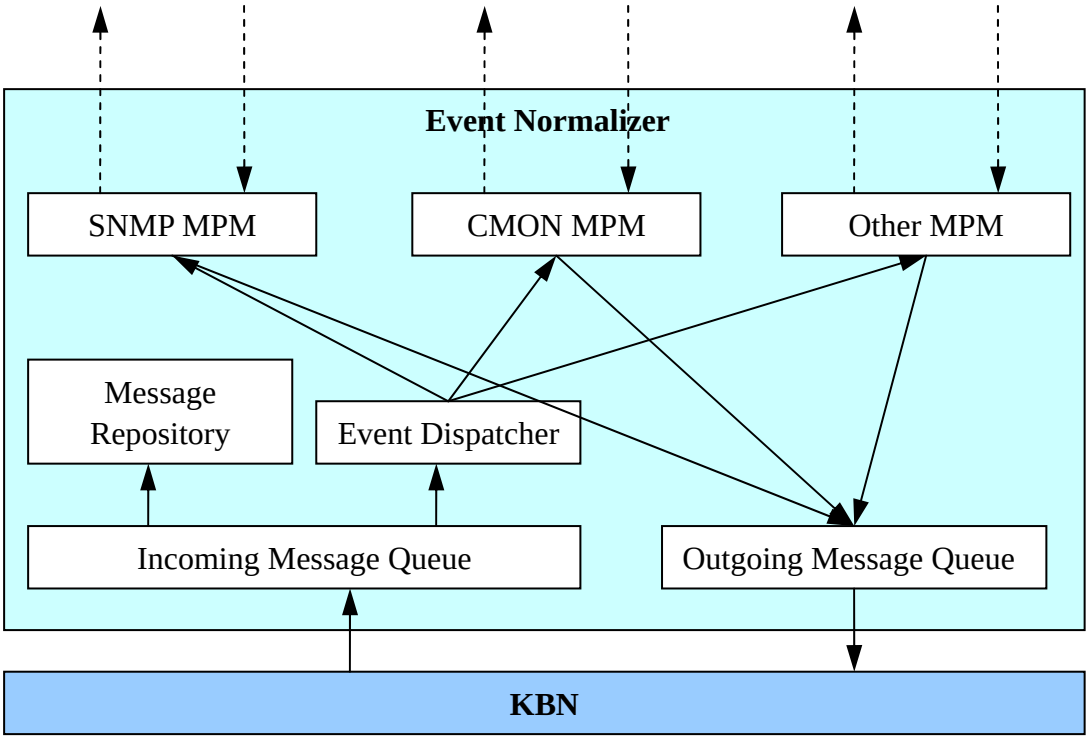
Figure 4-2 FMS Architecture

4.2.1.1 Event Normalizer

The Event Normalizer is the lowest components in the FMS architecture. It lies just above KBN and works as the adapter that between upper layer services and KBN. Its main task is to convert the KBN notification into upper layer services understandable message objects or operations and vice versa.

As illustrated by Figure 4-3, the Event Normalizer mainly contains 5 parts; they are Message Processing Model (MPM), Event Dispatcher, Incoming Message Queue, Outgoing Message Queue, and Message Repository. Every incoming message will be buffered in the Incoming Message Queue. The Event Dispatcher will then fetch the message from Incoming Message Queue and dispatch it to corresponding Message Processing Model according to the KBN tags carried by the message. The Message Processing Model is in charge of the encoding of message objects or service operations into network stream and the decoding of stream formatted messages into message objects or service operations. Every Message type that supported by this FMS should have a Message Processing Model (MPM) registered in the Event Normalizer. After being processed by Message Processing Model the outgoing message will then be buffered in the Outgoing Message Queue waiting for being pushed into KBN.

One thing worth mentioning is that some messages, such as SNMP trap message, can not be simply discarded after being processed by the Message Processing Model, for some information in the message could be used by other components such as trouble shooter in the future,. Therefore, these messages should be kept in the Message Repository, and for each message that is kept in the Message Repository, a number will be assigned to uniquely identify this message within the whole FMN. In order to avoid the infinite increasing on the size of this repository, a message collector will be running every given time period to collect the message that have not been use more than a given time. Instead of discarding the collected message, they will be kept in a message database for other use.



Message Objects/ Service Operations

Encoded Messages/ Notifications

Figure 4-3 Design of Event Normalizer

4.2.1.2 Knowledge Base

The Knowledge Base stores all the expert knowledge such as correlation rules, events, fault recovery policies, etc, in it, and provide access interface for upper layer services. It works like the “hard drive” of a computer and provides information for “processor” (upper layer services).

As illustrated by Figure 4-4 the Knowledge Base mainly consists of two parts: the knowledge files and the access interface. Knowledge files are the files that carry the expert knowledge, such as correlation rules, event information, and fault recovery policies and so on. Different type of expert knowledge will be stored using different formats in different knowledge files or event database. The proposed fault management system does not define the storage format for each type of expert

knowledge and it can be defined by the implementation. For example, in the evaluation implementation of the proposed fault management system, the correlation rules and event information are encoded using OWL and are kept in an ontology file.

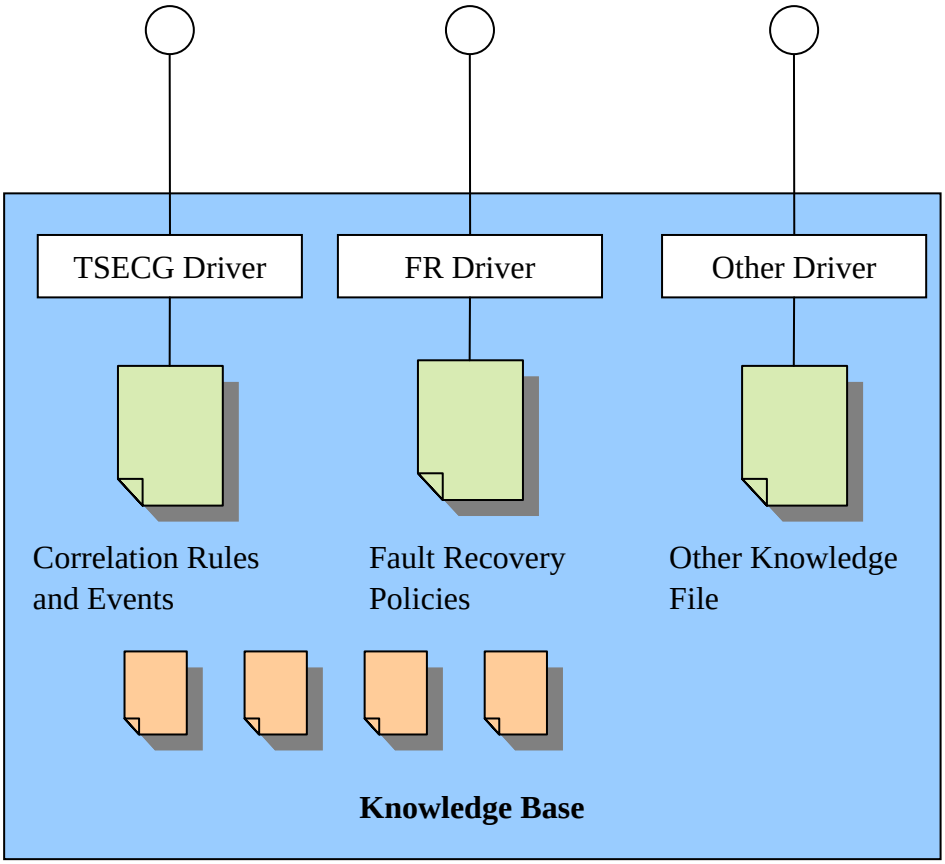


Figure 4-4 Design of Knowledge Base

Instead of defining the format of expert knowledge, a set of interfaces are defined for the access of expert knowledge. For each expert knowledge file that *will be directly used* by upper layer services, a driver class which implements the corresponding interface must be provided in order to enable the access to the expert knowledge stored in specific format. In the evaluation implementation of the proposed fault management system, a knowledge base access interface named as TopologySpecificEventCorrelationGraph is defined for the access to correlation rules and the information of events issued by the managed network, which is encoded using OWL in ontology file. By defining the access interface the implementation detail of

expert knowledge will become transparent to the upper layer service, thus making the update of knowledge base more flexible.

4.2.1.3 Service Organizer

The Service Organizer provides the execution environment, such as message queue and inter-service communication variables and so on, for fault management services. Furthermore, it will also chain the services it holds into a service chain to perform fault management. From the implementation point of view, the service organizer could be implemented as a process and the services could be implemented as the threads that spawned from the service organizer process.

Different services may require a different environment. Therefore, as an environment provider, the design of service organizer varies with the type of services it holds. However as an environment provider for fault management service, there is a common denominator for all the different designs. It has to provide the environment for at least two core services: Fault Correlator and Trouble Shooter. Figure 4-5 illustrates the common denominator design for the Service Organizer.

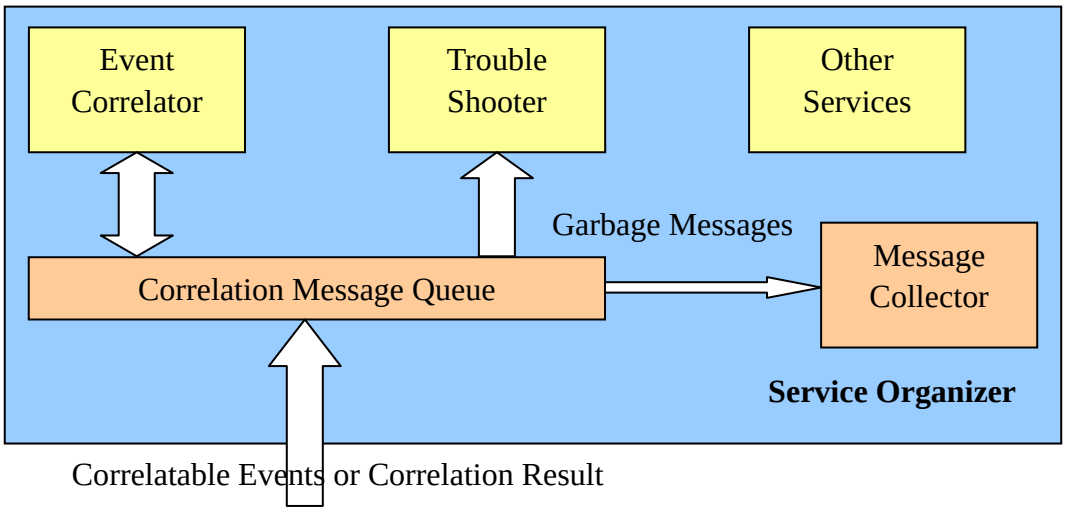


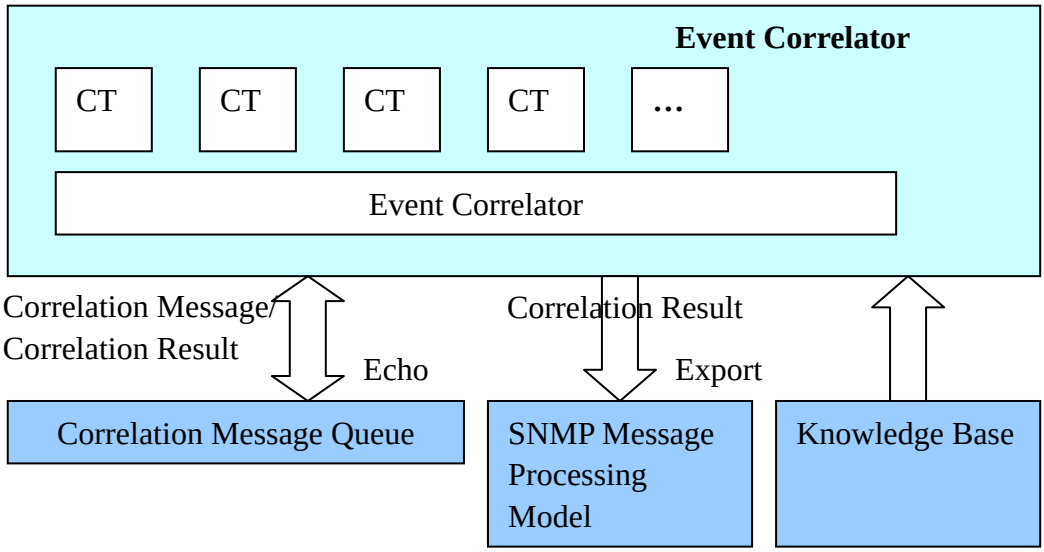
Figure 4-5 Common Denominator design of Service Organizer

As illustrated by Figure 4-5, the common denominator design of Service Organizer provides the running environment for two core services. The environment includes a Correlation Message Queue, where both the received *correlatable events* and *correlation results* will be kept. The Event Correlator will select the correlation candidates, e.g. correlation result and correlatable events from the correlation message queue, besides the Trouble Shooter will also select the correlation result that should be addressed from the correlation message queue to perform fault recovery operation. To avoid the infinite expansion of the correlation message queue, the Message Collector is provided by Service Organizer to maintain the Correlation message queue by removing the messages that are regarded as garbage messages. The criteria that whether messages in the correlation message queue can be regarded as garbage message are defined by policies. A very important criterion for the garbage message for example would be whether the occurrence time of this message has exceeded a given time period.

4.2.1.4 Event Correlator

Event Correlator is one of the core services that will on the service organizer. It will provide the event correlation service for the fault management, which is the prerequisite service for trouble shooting.

The proposed fault management system does not provide a unified interface for the creation of services. Therefore, in order to add a new service, for example, fault logger, into the service organizer given by Figure 4-5, the whole service organizer has to be updated from code level to hold the new service and to provide the running environment for it. Figure 4-6 shows the design of event correlator and its dependent components.



CT: Correlation Thread

Figure 4-6 Design of Event Correlator Service and its dependent components

As illustrated by Figure 4-6 Event Correlator service contains two main parts: Event Correlator and Correlation Thread. The main tasks of Event Correlator are to analysis the messages in the Correlation Message Queue, get the correlation tasks and then assign the tasks to the Correlation Thread. The Correlation Threads, which are the threads spawned from Event Correlator and implement a distributed correlation scheme, will accept the task assigned by Event Correlator and then perform correlation on it.

After the Correlation Thread finishes its correlation task, the correlation result will be either echoed into Correlation Message Queue if it could be used by the other correlation in this FMS, or be exported into KBN if it could be used by correlation in other FMS. We will come to the distributed correlation scheme later on in section 4.3.

4.2.1.5 Trouble Shooter

As with the Event Correlator, Trouble Shooter is one of the core services in the proposed fault management system. It will generate the fault recovery operations according to the fault recovery policies stored in the knowledge base using the

correlation results.

A correlation result can either be used by correlator to perform further correlation or be used by trouble shooter to perform trouble shooting, thus a method should be used to address this conflict. In this current design, a timer will be used to address this problem. For the further correlation has higher priority than trouble shooting (If a correlation result could be used for other correlation, that means an even bigger problem needs to be correlated using this correlation result, which obviously has higher priority than just solve this minor problem.), the trouble shooter can not use a correlation result to perform trouble shooting process until the timer for this correlation result expired. Once a correlation result is selected to perform trouble shooting process, the trouble shooter will inform the sender of this correlation result that this correlation result can be addressed (The correlation result only carries the information for correlation, which is not enough for trouble shooting. Thus the fault recovery operation should be generated and performed in the FMS that originally receive the events, e.g. SNMP trap, from managed network). Because the correlation result is generated hierarchically from a low level event correlator to a high level event correlator, this process will be performed reverse to the correlation process from high level trouble shooter to low level trouble shooter.

As mentioned before, the fault recovery policies are stored in Knowledge Base using dedicated files. Therefore, the fault recovery operation should access the Knowledge Base via its access driver. As they are kept in a file, the policies can easily be updated or replaced, which increases the flexibility of the proposed fault management system.

It should be noted that trouble shooter of the proposed fault management system has not been well defined yet, and further definition should be considered future work.

4.2.1.6 Configuration Manager

Unlike the service organizer, the configuration manager is a unified name for all the

configuration management modules such as Knowledge base update module, subscription manager and so on. There is not a concrete module or subsystem named as configuration manager.

A FMS can have multiple Configuration Managers to perform different configurations of the FMS. However, the Subscription Manager is a compulsory Configuration Manager, and it has to be implemented by all implementation of the proposed fault management system.

4.2.1.7 Subscription Manager

Subscription Manager is a compulsory Configuration Manager for the proposed fault management system. It controls all the messages including correlation events, inter-FMS communication messages and so on that can be received by the FMS. The subscription manager performs its task by undertaking subscribe/unsubscribe operations of its interests upon the KBN. Figure 4-7 is an example of how subscription manager controls the correlation task of a correlator.

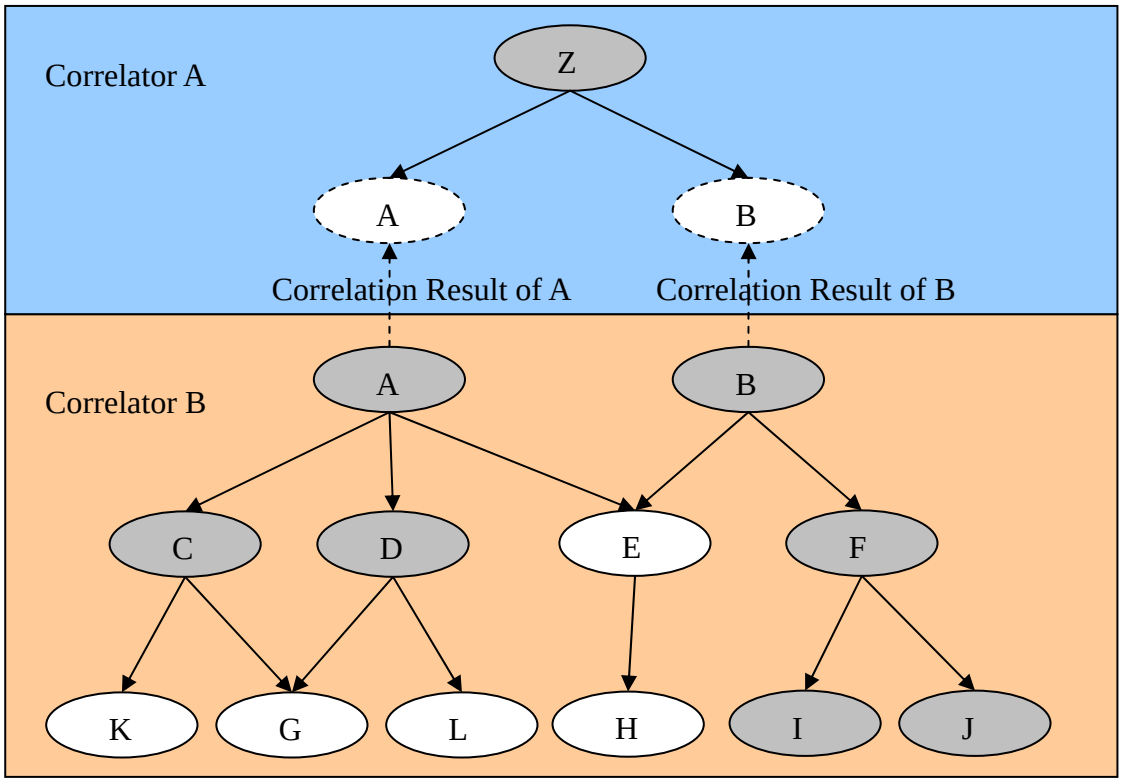


Figure 4-7 Subscription manager controls the correlation task

As illustrated in Figure 4-7, Correlator A will perform correlation over event A and correlation result (A, B), which is provided correlator B; The correlator B will perform correlation over event set (A, B, C, D, E, F, G, H, I, J, K, L), and then provide the correlation result of event A and B to correlator A. In order to have correlator A and correlator B perform the above mentioned correlation task, the subscription manager of correlator A and B will put subscriptions like those in Figure 4-8 to KBN node.

```
{(event @= Z), (message_type=correlatable_event)}
{(event@=A), (message_type=correlation_result)}
{(event@=B), (message_type=correlation_result)}
```

(a) Subscriptions put by correlator A

```
{(event@>A), (message_type=correlatable_event)}
{(event@>B), (message_type=correlatable_event)}
```

(b) Subscriptions put by correlator B

Figure 4-8 Subscriptions of correlator A and correlator B

The subscription can be either input to the KBN by a system operator from the user interface or read from a configuration file. All the subscriptions for different messages, e.g. correlation message, fault recovery message, and so on, will be kept in a vector. A set of operations such as subscribe, unsubscribe and modify subscription and so on will be provided by the subscription manager to the on-the-fly subscription management.

A more automatic subscription manager which provides the support to Plug and Play (PnP) FMS could be designed by adding in a new configuration manager that can listen both for the addition and removal of a FMS through a heartbeat beacon, and

then automatically adjusts its subscription according to the subscription of the newly added or removed FMS.

The design of subscription manager has not been fully finished as yet, and in the current evaluation implementation the subscription manager can only control its interests by reading subscriptions from a configuration file.

4.3 Distributed Correlation Scheme

4.3.1 Scheme background

The correlation scheme proposed in the project is a pure distributed scheme. Current distributed correlation schemes, such as the one used in Madeira project [18, 19], perform distributed correlation by applying current centralized correlation scheme at different level of the managed network or different network domain resided at different geographical location. By doing so, each Network Element in the managed network needs to be explicitly configured so that all its fault events can be routed to the right FMS; or, all the fault events are forwarded to the FMS using broadcast in the network. There are explicitly drawbacks in those two aforementioned forwarding schemes: either there is a tight coupling between the managed network and specific FMS (explicitly configuration) or there is high possibility to cause event storms (broadcasting).

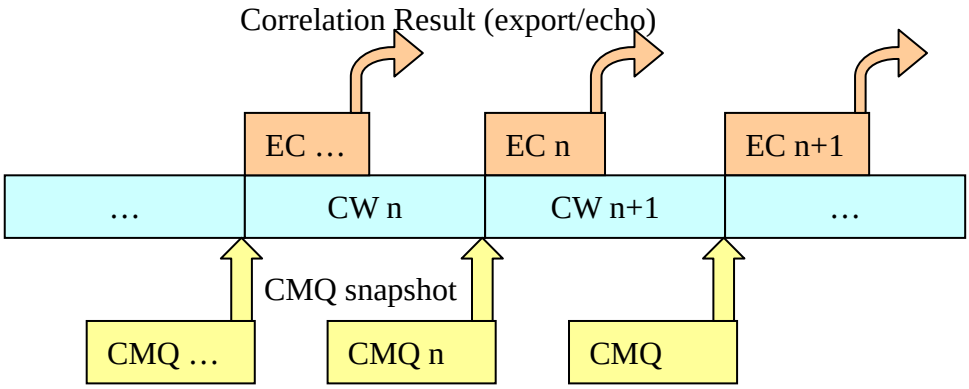
The proposed distributed correlation scheme distributes a single correlation task among the whole FMN, and each FMS within the FMN will take part of the correlation. The low level correlator will provide the correlation result for high level correlation, and the whole correlation task for the managed network will then be performed hierarchically. There is neither tight couple between managed network and specific FMS nor the high possibility of causing an event storm. All fault events will be pushed into the FMN which is based on KBN, and then be routed in the FMN

according to the subscription of specific FMSs. By adding the PnP feature into FMS, the failure in one specific FMS can not disable the whole fault management system. Another FMS will assume the correlation task of the failed FMS by adjusting its subscription and have all the events that are originally forwarded to the failed FMS re-route to this FMS. By doing so, rather than disable the whole fault management system, the failure of a FMS will only degrade slightly the performance of the proposed fault management system.

Besides this pure distributed feature, the proposed correlation scheme can also perform the correlation under the circumstance where some correlation symptoms are lost. When performing the correlation under this circumstance, several factors with given weight will be used to evaluate the candidate correlation results.

4.3.2 Scheme Overview

The whole running process of the event correlator will be divided into time scales. Each time scale is called a *correlation window*. The length of a correlation window is adjustable. Usually a high level correlator will have a longer correlation window than a low level correlator. Event correlation will be started immediately after a correlation window ends. Figure 4-9 gives an overview to the event correlation process.



EC: Event Correlation
CW: Correlation Window
CMQ: Correlation Message Queue

Figure 4-9 Overview of event correlation

Once the correlation has started, the event correlator will first select correlation tasks from the correlation message queue using Correlation Task Analysis which will be introduced in the next section. The selected correlation tasks will then be assigned to the correlation threads. Correlation Thread will then perform event correlation using the proposed event correlation scheme. After correlation, the correlation result will be either exported to a higher level correlator or echoed back into the correlation message queue. In the next section, a detailed description will be given to this correlation scheme.

4.3.3 Technology Background and Terminology Definition

As a new correlation scheme, some new terminologies are defined in order to fully and clearly describe this correlation schemes. Besides some new calculation rules are defined to support the proposed correlation scheme. In this section, these definition and calculation rules will be introduced.

- Event, Observable Event and Non-Observable Event

An event represents the occurrence of something in the managed network. An event

could indicate the occurrence of a severe fault or simply a change of configuration in the managed network. Some events have a dedicated message sent out when they occurred; these events are called *Observable Event*. Some events have no dedicated message to indicate their occurrence; these events are called *Non-Observable Events*. The occurrence of non-observable events can only be deduced from the occurrence of other events that are caused by the non-observable events. For example, in the real-world scenario modeled in Figure 3-3, event XCRuined_PE1 is a non-observable event, and its occurrence state can only be deduced from the occurrence of LspBroken_PE1_PE2.

- Event Correlation Graph

The Event Correlation Graph models the managed network from the events and their relationships point of view. It represents symptom/cause relationships between events that issue from the managed network. These expert knowledge will be used in the event routing in KBN and subscription management in FMS. Figure 4-10 is an event correlation graph with 13 events.

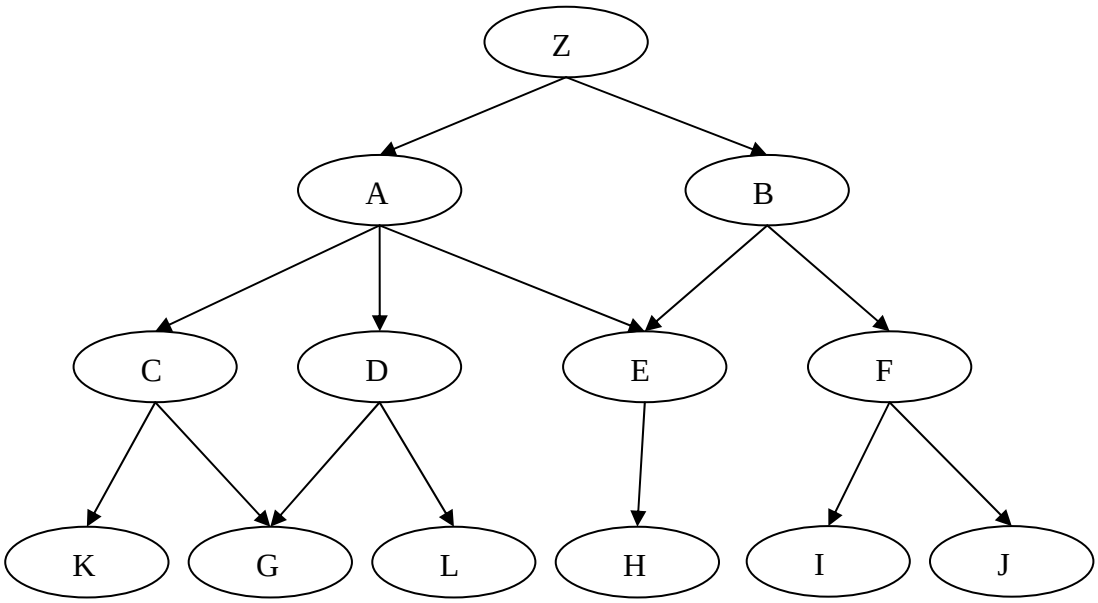


Figure 4-10 Example Event Correlation Graph

As illustrated by Figure 4-10, the Event Correlation Graph is a Direct Acyclic Graph

(DAG). Every Node represents an event and the arrow between two nodes represents the correlation relationship, for example, C can correlation to K, C can correlation G, and so on.

Event Correlation Graph constructed in this project is similar in shape and semantics with those used in [4, 8]. However, instead of being used directly for event correlation Event Correlation Graph in this project is used for event routing in KBN and subscription management in FMS. As mentioned in section 3.2.1, the Event Correlation Graph in the implemented fault management system will be modeled using OWL.

- Symptom and Cause

If the occurrence of event A causes the occurrence of event B, we will say “event B is the *symptom* of event A”, or “event A is the *cause* of event B”. The symptom/cause relationship is not absolute. An event(s) can be the cause of another event(s), while it can be the symptom of a third event(s).

- Root Cause, Global Root Cause and Local Root Cause

Root Causes are causes that have no further causes. That is to say, a root cause can not be the symptom of another event. They are usually the where the fault is and what we need as the output of the correlation. For example, events K, G, L H, I and J are the root causes of the Event Correlation Graph given by Figure 4-10

We only refer to the phrases *Global Root Cause* and *Local Root Cause* when calculating the Direct Correlation Rule and calculating Correlation Table, which we will refer to in the following text. When an Event Correlation Graph is spread over several FMSs, each FMS will hold part of the Event Correlation Graph (this will be done through subscription manager). If an event is a Local Root Cause of an FMS, then the event is root cause within the Event Correlation Graph that FMS holds; if an event is a Global Root Cause, then this event is the root cause of the whole Event

Correlation Graph.

- Correlation Rule Set

Correlation Rule is the text representation of Event Correlation Graph. It has more semantics than the Event Correlation Graph. The Event Correlation Graph can not represent the “and” relationship, namely, there is no way to distinguish “A can correlate to C and D” and “A can correlate to C or D” in the Event Correlation Graph. However, this can be easily represented using correlation rule “A->C∧D”. Therefore, the event correlator in the proposed fault management system uses Correlation Rule Set instead of Event Correlation to perform event correlation, and Event Correlation Graph will be used only for Correlation Task Analysis, Subscription Management and Event Routing. Figure 4-11 is the Correlation Rule Set that generated from Event Correlation Graph given by Figure 4-10.

Z -> A∧B
A -> C∧D, A -> E
B -> E, B -> F
E -> H
F -> I∧J
C -> K, C -> G
D -> L, D -> G

Figure 4-11 Correlation Rule Set representation of Figure 4-10

“A -> C∧D” is a correlation rule, which means “A can correlate to C and D”. The arrow “->” is the correlation operation which means “can correlate to”. Terms on the left of the arrow are the symptom of this correlation rule, and terms on the right of the arrow are the cause of this correlation rule. The symptom and cause are composed of events that relate to each other using either conjunction “∧” or disjunction “∨”. New rules can be generated by calculating old rules. There are some calculation rules (these “rules” are not correlation rules; they are the “rules” in mathematics) for the correlation rules, which we will refer to later on in the following section.

- Correlation Event Level

Correlation Event Level is the level that this event locates in the Event Correlation Graph, the calculation of Correlation Event Level is given below:

$$lvl(A) = \begin{cases} 0 & if\ A \in GlobalRootCause \\ \max(lvl(b)) + 1 & otherwise, b \in cause(A) \end{cases}$$

According to this correlation event level calculation equation, we can get that in Figure 4-10 *lvl(A)=2, lvl(C)=1* and *lvl(K)=0*.

- Correlation Depth

The correlation depth of a correlation rule is the level that this correlation rule covers. We assume that R represents a correlation rule; sympt(R) represents the symptom of correlation rule R; cause(R) represents the cause of this correlation rule. The calculation of Correlation Depth is given below:

$$dpth(R) = \max(lvl(sympt(R))) - \min(lvl(cause(R)))$$

According to this correlation depth calculation equation, *dpth(A->C∧D)=1* and *dpth(A->(C∧D)∨H)=2*.

- Basic Correlation Rule Set

The correlation rule set directly generated from Event Correlation Graph are called *base correlation rule set*. All rules in the base rule set should be *atomic*, namely, they are not generated by the calculation of other rules, and besides their correlation depth must be 1. Figure 4-11 is a base correlation rule set. The Basic Correlation Rule Set will be stored in knowledge base and be accessed using the access interface provided by knowledge base.

- Direct Correlation Rule

A *Direct Correlation Rule (DCR)* is the correlation rule whose cause is composed of root causes (either local root causes or global root causes). It is named as direct

correlation rule because it can be directly used for correlation. The Direct Correlation Rule can be gained by calculating the rules in the base correlation rule set. For example in Figure 4-11, the DCR for event A is A->(K∧L)∨G∨H, and the DCR for event B is B->(I∧J)∨H.

- Correlation Branch

The correlation branch of an event is a set of events that contains all the events that can cause this event, including both root causes and intermediate events. For example, in Figure 4-10, the Correlation Branch for event A is {A, C, D, K, L, G, E, H}, and the Correlation Branch for event B is {B, F, I, J, E, H}.

- Event Branch Radius

The Event Branch Radius indicates the time period from the time when the root event occurs to the time when the last event in the branch set is received. This value relate to the size of branch set and the underlying network delay. For example, in Figure 4-10, under the normal condition the Branch Radius of branch set {K, C} is the time that K occurs to the last event (maybe K, maybe C) is received by a Correlator. The branch set {K, C} should have a lower radius than the branch set {K, C, A} for the size of former one is small than the later one.

- Correlation Rule calculation rules

Two calculation rules are defined for the calculation of Correlation Rules. The whole correlation schemes are constructed based on these two rules.

Merging Rule: Given correlation rule R1 and R2, let *R3=merge(R1,R2)*, then:

$$R3 = \begin{cases} sympt(R1) -> cause(R1) \vee cause(R2) & if\ sympt(R1) = sympt(R2) \\ sympt(R1) \wedge sympt(R2) -> cause(R1) \wedge cause(R2) & otherwise \end{cases}$$

Substitution Rule: Given correlation rule R1 and R2, R1 can be substituted by R2, iff $\exists e \in cause(R1), sympt(R2) = e$. If R1 can be substituted by R2, let

R1=sympt(R1)->(sympt(R2)∧term1∧…) ∨(term2∧term3∧…) ∨(⋯).

Then after substitution

R1=sympt(R1)-> (cause(R2)∧term1∧…) ∨(term2∧term3∧…) ∨(⋯).

After the substitution, the *cause(R1)* needs to be normalized to the "and or" format.

Figure 4-12 gives an example of how the merging rule and substitution rule will be used in rule calculation. It calculates the DCR of event A in Figure 4-11.

1. Merge A -> C∧D and A -> E:
A -> (C∧D)∨E = A -> (C∨E)∧(D∨E)
2. Merge C -> K and C -> G:
C -> K∨G
3. Merge D -> L and D -> G:
D -> L∨G
4. Substitute A -> (C∨E)∧(D∨E) by C -> K∨G and D -> L∨G
A -> (K∨G∨E)∧(L∨G∨E)
5. Substitute A -> (C∨E)∧(D∨E) by E -> H
A -> (K∨G∨H)∧(L∨G∨H)
6. Normalize, applying distribution rules to causes, we can get:
A -> G∨H∨(K∧L)

Figure 4-12 DCR calculation using merging rule and substitution rule

4.3.4 Correlation Task Analysis

Correlation Task Analysis is the operation that selects correlation task from correlation message queue. It is the first operation performed by event correlator immediately after the correlation window ends.

The correlation message queue can still be modified by other components such as Message Processing Model and so on. Therefore, in order to keep the consistency of the event set of Correlation Task Analysis. A snapshot of the correlation message queue will be taken immediately after the correlation window ends and the Correlation Task Analysis will be performed on the "snapshot" of the correlation message queue.

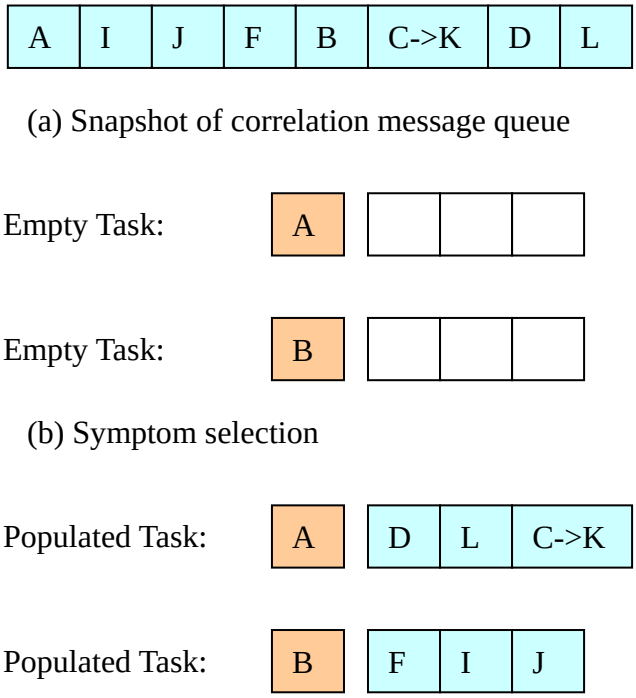


Figure 4-13 Correlation Task Analysis

We assume that a snapshot of the correlation message queue is given by Figure 4-13 (a) and the correlatable events and correlation results in this snapshot are based on Event Correlation Graph given by Figure 4-10.

When the event correlator starts the Correlation Task Analysis, it will first select the symptoms for correlation task. In this step, only *correlatable event* will be considered. The symptom selection process will use the Event Correlation Graph and only the symptom that can not be covered by other correlatable events in this snapshot will be selected as the symptom. In Figure 4-13 (a), two symptoms will be selected: event A and event B. Then for each selected symptom, an empty task will be created (Figure 4-13 (b)). Second, the empty correlation task will be populated by the correlatable events or correlations results in the snapshot. This step will be performed by comparing the correlation branch of the selected symptoms with elements in the

snapshot, and then populating the empty task with the correlatable event/ correlation result that both in the correlation branch and the snapshot. For example, the correlation branch for symptom A is {A, C, D, E, K, G, L, H}. After comparing the received elements in the snapshot with the correlation branch of A, the empty correlation task of A will be populated with event/correlation result (C->K, D, L), and so it is the same with the correlation task of B. The outcome correlation tasks of the Correlation Task Analysis are given by Figure 4-13 (c).

After Correlation Task Analysis, each generated correlation tasks on which the correlation will be carried out will then be assigned to a correlation thread.

4.3.5 Correlation Schemes

This correlation scheme uses the calculation of *correlation table* to perform correlation. Three *weighted belief factors* will be used to measure the belief degree of a correlation result. Because of the introduction of correlation table and weighted belief factor, this correlation scheme will treat the element missing correlation and normal correlation the same way.

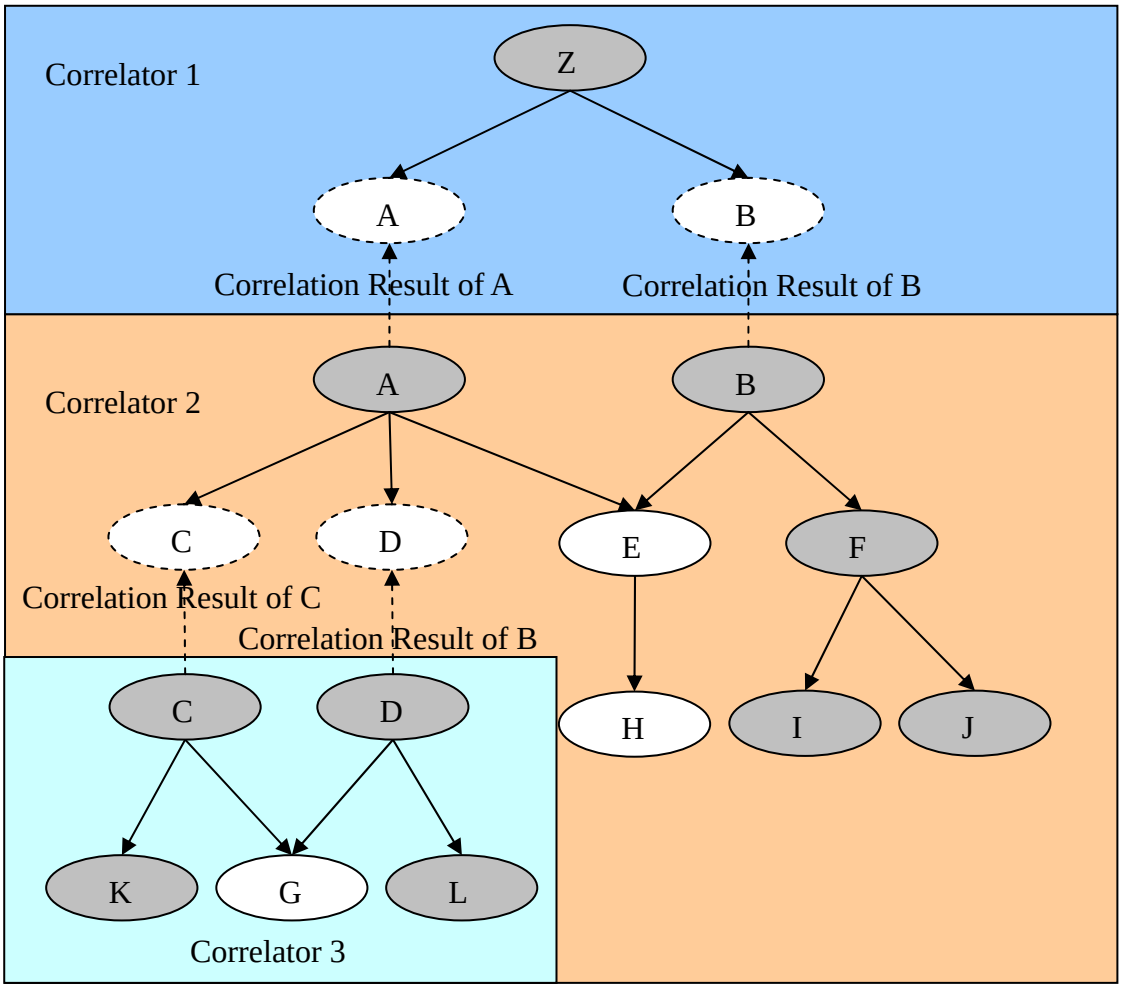


Figure 4-14 Three correlators example

The whole correlation calculation scheme can mainly be divided into the following 5 steps: DCR calculation, correlation table initialization, delay correlation, event guessing, and post-correlation processing. In the remainder of this section, a detailed description will be given for each of these steps, using the example provided in Figure 4-14 to aid the description.

As illustrated in Figure 4-14, there are three FMS running in the FMN. There is a correlator running on each FMS and each correlator takes the responsibility for the correlation of part of the whole Event Correlator Graph, which is illustrated in Figure 4-14 by boxes of different colors. The circles are the events that this correlator interests and the dash circles are the correlation results that this correlator interests. The circles that are marked in grey represent that these events are received, and

circles that are marked in white represent that these events have not been received yet or they are non-observable events.

We will mainly concentrate on the correlation process of correlator 2 hereafter, and the following assumptions will be made for correlator 2:

- The export event set is {A, B}, which means the correlation result of symptom A and B will be exported into KBN to higher level correlator.
- The local root cause set is {C, D, H, I, J}.
- The correlation result of event C, and D has already been received before correlation window of correlator 2 ends.
- Two correlation threads, say Thread-1 and Thread-2, are running on correlator 2, and Thread-1 is responsible for the correlation task {A, C->K, D->L}, and Thread-2 is responsible for the correlation task {B, F, I, J}.
- All events except G are observable events.
- The base correlation rule set is given in Figure 4-11.

These assumptions are not fixed, and they may be changed during the following description, but I will inform explicitly if assumptions has been changed.

4.3.5.1 Direct Correlation Rule calculation

Direct Correlation Rule calculation is the first step after the correlation thread started. It calculates the DCR for the symptom of the correlation task (In Thread-1 the symptom of its correlation task is event A, and in Thread-2 the symptom of its correlation task is event B) by merging and substituting rules in the base correlation rule set with each other. The DCR calculation process will stop when *all the events in its cause are in local root cause set*, and then take the calculation output as the DCR of an event. An example calculation process has already been given in Figure 4-12 and here we will not give this process again. The DCR for event A and B are:

A -> H∨(C∧D)

B -> H∨(I∧J)

4.3.5.2 Correlation Table Initialization

As mentioned before, the whole correlation process is performed by calculating the correlation table. This step will initialize the correlation table used in the future calculations.

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements
A->H	1	1	2	H	H	A
A->C∧D	0	0	0	None	None	A, C, D

Table 4-1 Initial Correlation Table for Thread-1

As illustrated in Table 4-1, each correlation table contains 7 columns, and they are: Potential Result, Lost Number, Guess Number, Mis-match Number, Used Elements, Guessed Elements and Lost Elements. The *Potential Result* column contains the correlation equations that have the potential of being the correlation result. The value of this column is gained from the DCR generated in the last step. All terms that are in the cause of the DCR and connected using “or” operator will be considered as potential results. For example, in Thread-1, there are totally two potential results: A->H and A->C∧D, which are generated from the DCR of Thread-1: A -> H∨(C∧D).

The *Lost Number* column indicates the number of lost events/correlation results (notice: only observable events are considered) for this potential result, and these lost elements will be kept in the column *Lost Elements*. These two columns will be filled in through comparing the required elements for this potential result with the elements in the assigned task. For example, in Thread-1, the correlation task is {A, C->K, D->L}, and the required elements for potential result A->H is {A, H}, so the lost element is H, and the lost number should be 1. For potential result A->C∧D, the required elements are {A, C, D}, so there is no lost element and then the lost number

should be 0.

Similar to Lost Elements and Lost Number, the *Guessed Elements* and *Guessed Number* indicate the elements (notice: both observable and non-observable events will all be considered) that need to be guessed for this potential result and its number. This is also performed by comparing the required elements for this potential result with the correlation task assigned. Elements that are required by this potential result but did not received in the correlation task will be regard as *Guessed Elements*. For example, in Thread-1, for potential result A->H, the guessed elements should be H and the guessed number should be 1, and for potential result A->C∧D, there is no guessed element and therefore the guessed number is 0. It seems that the Lost Elements and Lost Number will always hold the same value with the Guessed Elements and Guessed Number. The main difference between the Lost Elements/ Lost Number and the Guessed Elements/ Guess Number is that Lost Elements/ Lost Number do not consider the Non-observable Event but the Guessed Elements/ Guessed Number do.

The Used Elements column indicates the elements that have been used for the potential result. The elements in the Used Elements are the elements that exist both in the required elements for the potential result and in the correlation task. For example, the used elements for potential result A->H is A, and the used elements for potential result A->C∧D are {A, C, D}.

The Mis-match Number is the number that the required elements fail to match the correlation task. For example, the mis-match number for potential result A->H is 2, and the mis-match elements are C and D; the mis-match number potential result A->C∧D is 0 because all required elements of A->C∧D match the correlation task.

Notice should be paid that, in practice, the event itself will not be kept in the Lost Elements list, Guessed Elements list or Used Elements list. The elements that kept in the Lost Elements, Guessed Elements and Used Elements are only the identity

numbers that globally uniquely identify these elements.

The Lost Number, Guessed Number, and Mis-match number are the three selected weighted belief factors for a potential correlation result. They are selected as the belief factors for the reason that they represents the event matching degree of the correlation task to the potential correlation result from different aspects separately. Lost Number represents the how many observable events are missing is this potential correlation result is selected as the correlation result, the Guessed Number supplements the Lost Number with non-observable result, and the Mis-matching Number represents how many event in correlation task will not be used if this potential correlation result is selected as the final correlation result. Different weights will be assigned to these belief factors:

weight(Lost Number)=2;
weight(Guessed Number)=1;
weight(Mis-matching Number)=3;

Currently there are no well-defined criteria for determining the belief factors and setting the weight for them, and thus currently selected belief factors can not fully represent the belief degree of the correlation task to a potential correlation result. However, for the currently selected belief factors mainly determine the belief degree from event matching degree point of view, they can represent the belief degree of this potential result to some extent. The identification of criteria for determining the belief factor and their weight setting has been left for future work. Table 4-2 gives the initial correlation table for Thread-2.

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements
B->H	1	1	3	H	H	B

B->I∧J	0	0	0	None	None	B, I, J, F

Table 4-2 Initial Correlation Table for Thread-2

4.3.5.3 Delay Correlation

Delay Correlation will be performed when there is no Correlation Table Entry in the initial correlation table. It will delay the whole correlation process by a given time period, waiting for the guessed element.

The Delay Correlation will be performed in three steps. First, a *listening list* will be calculated by aggregating all the guessed elements in all the correlation table entry of initial correlation table. Second, the correlation thread will wait for a given time period waiting for the receiving of events or correlation results listed in the listening list. The waiting time period is called a *delay window*. The length of the delay window is configurable. Third, after the delay window expired, the correlation thread will then detect the Correlation Message Queue whether there are events/correlation results that listed in the listening list received. If there are, the correlation thread will go fetching these events/correlation results, and add these newly received events/correlation results into the its correlation task and then re-initialize the correlation table.

An example will be used to explain these three steps. In order to explain this better, we will slightly change the assumption made for correlator 2. Instead of being received before the correlation window ends, we assume that only the correlation result of D has been received before the correlation window ends and the correlation result of C is received soon after the correlation window of correlation 2 ends (within the delay window). Then the correlation task of Thread-1 is {A, D->L} and the initial correlation table will become the one given by Table 4-3.

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements

A->H	1	1	2	H	H	A
A->C∧D	1	1	0	C	C	A, D

Table 4-3 Initial Correlation Table for Thread-1 when C is received after correlation window

According to what mentioned earlier in this section, the delay correlation will be performed when no correlation table entry has its Guessed Number as 0. Therefore the delay correlation for Thread-1 (Table 4-3) will be performed. The listening list is for this delay is {H, C}, and we assume that the delay window is 1000ms. After delay window ends, Thread-1 will detect that the correlation result of C has already been received in the correlation message queue. It will then add the correlation result of C into the correlation task, and the correlation task for Thread-1 will become {A, C->K, D->L}. After these, the correlation table will be re-initialized using the new correlation task, and the correlation table will become the one given in Table 4-1.

The delay correlation will only be performed once during the whole correlation process for a correlation task. A more detailed description on the necessity of delay correlation and the setting of delay window will be given in section 4.3.6.

4.3.5.4 Correlation Result Merging

After the first three steps, an initial correlation table has already been generated. However, the causes of the potential correlation results in the initial correlation table are all local root causes, so they can not be used as the final correlation result. In order to gain the final correlation result, we have to substitute the local root causes in the potential correlation results with the correlation results received from low level correlator. This merging process includes not only the substitution of two correlation equation, but also the *merging of other columns* of two correlation table.

In order to clearly explain correlation result merging process, we will first enrich the

example we are using. A new assumption is made here:

- Event K lost on its way to correlator 3, so the correlation result of C is a guessing correlation result. The correlation result of event C and D is given in Table 4-4 and Table 4-5.

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements
C->K	1	1	0	K	K	C
C->G	0	1	0	None	G	C

Table 4-4 Correlation Table in Correlation Result of C

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements
D->L	0	0	0	None	None	D, L

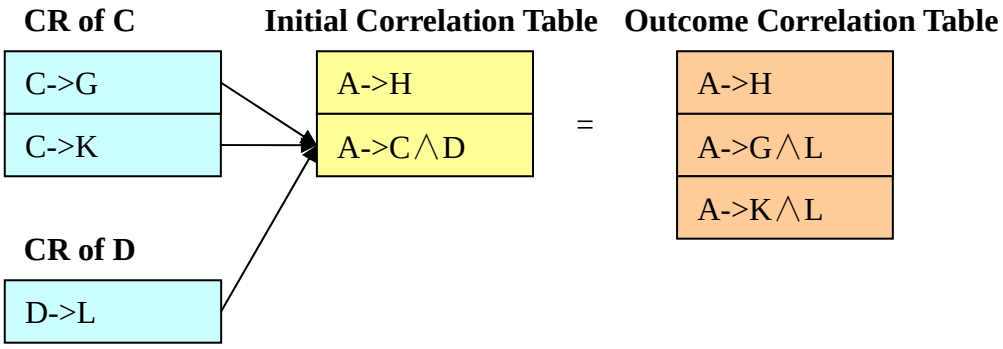
Table 4-5 Correlation Table in Correlation Result of D

The merging process will be performed by merging the initial correlation table by entries from the correlation tables of the received correlation results one by one. The substitution algorithm is given by pseudo-code in Figure 4-15, and in Figure 4-16 an example is given illustrates this process:

```
output_correlation_table = new correlation_table();
for(dst_entry in init_correlation_table){
    tmp_table = new correlation_table();
    tmp_table.add(dst_entry);
    for(tmp_entry in tmp_table){
        for(correlation_result in correlation_task){
            if(canBeSubstituteBy(tmp_entry, correlation_result)){
                received_correlation_table = getCorrelationTable(correlation_result);
```

```
                    // merge the entry in initial correlation table with received
                    //correlation table
                    merged_table = merge(tmp_entry, received_correlation_table);
                    // replace the entry with the newly merged table
                    tmp_table.replace(tmp_entry, merged_table);
                    // reset tmp_entry, iterate from the start of tmp_table again
                    tmp_entry.reset();
                    // reset correlation_result
                    correlation_result.reset();
                }
            }
        }
        output_correlation_table.addAll(tmp_table);
}
```

Figure 4-15 Correlation Result Substitution algorithm



CR: Correlation Result

Figure 4-16 Correlation Result merging process

According to the algorithm given in Figure 4-15, this process will not stop until there is no more correlation table entry in the initial correlation table can be substituted.

The merge() function in Figure 4-16 is the core function of the merging process. It

merges the correlation table entry in original correlation table, for example, dst_entry, with the correlation table in the received correlation results, for example, src_table. Each entry in the src_table will be merged with the dst_entry column by column. The outcome of this merging is a new correlation table with the same size with src_table. A description to the merging of different column will be given in the remaining text of this section.

The merge of potential result column is a simple substitution processing. The potential result of dst_entry will be substituted by the potential result of each entry in src_table. For example, A->C∧D will be substitute by two potential result C->G and C->K, thus generate two new entry with A->G∧D and A->K∧D as their potential result respectively.

The merge of guessed element, used element, and lost elements are simply merging these three element sets respectively. The new Guessed Number and Lost Number will gained by simply getting the size of the respective newly merged element set. The new Mis-matching Number equals the difference between the size of merged correlation task set and the size of merged used elements. If this different is negative, then the Mis-matching will be set to 0. For example, for the merging of A->C∧D with C->K, the merged Guessed Elements is {K}; the merged Lost Elements is {K}; the merged Used Elements is {A, C, D}; the new Guessed Number is 1; the new Lost Number is 1; as to the new mis-matching number, the merged correlation task is {A, C, D}, and the merged Used Elements is {A, C, D}, so the merged mis-matching number should be 0. Table 4-6 lists the new merged correlation tables for Thread-1 and because there is no received correlation result for Thread-2, therefore correlation table for Thread-2 is the same with Table 4-2.

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements

A->H	1	1	3	H	H	A
A->G∧L	1	1	0	G	C	A,C,D,L
A->K∧L	1	1	0	K	K	A,C,D,L

Table 4-6 Correlation Table for Thread-1 after Correlation Result Merging

4.3.5.5 Event Guessing

After Correlation Result Merging if there is still potential result whose causes are not the global root cause, the event guessing process will then start. The main idea of Event Guessing process is to generate a guessing correlation result for each non-global root cause in the potential result under the assumption that no event has been received, and then merge each guessing correlation result with the outcome correlation table of Correlation Result Merging.

In order to explain this clearer, assumption will be made that the correlation result for C lost during its way to Correlator 2. Therefore, after Correlation Result Merging the Correlation Table should be as Table 4-7.

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements
A->H	1	1	2	H	H	A
A->C∧L	1	1	0	C	C	A, D, L

Table 4-7 Correlation Table for Thread-1 after Correlation Result Merging (Correlation Result of C lost)

In the correlation table shown in Table 4-7, there is one event, e.g. C, in potential result A->C∧L does not belong to global root cause, and then the event guess process will start. First, the guessing correlation result for event C will be generated under the assumption that no event has been received (Table 4-8).

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements
C->K	2	2	0	C, K	C, K	None
C->G	1	2	0	C	C, G	None

Table 4-8 Guessing Correlation Result for event C

After generating the guessing correlation result, all guessing correlation results (Table 4-8) will then be merged with the outcome correlation table generated from Correlation Result Merging (Table 4-7) using the same method introduced in section 4.3.5.4. The result correlation table is given in Table 4-9.

Potential Result	Lost Number	Guessed Number	Mis-match Number	Lost Elements	Guessed Elements	Used Elements
A->H	1	1	2	H	H	A
A->G∧L	1	2	0	C	C, G	A, D, L
A->K∧L	2	2	0	C, K	C, K	A, D, L

Table 4-9 Correlation Table for Tread-1 after Event Guessing process

4.3.5.6 Post-correlation processing

The Post-correlation processing mainly includes the following operations: removing unsupported correlation result, generating correlation result and echoing/exporting correlation result.

After the correlation table is generated from the event guessing process, all entries that are unsupported for this correlation will be removed. For example, in Table 4-9 the entry A->G∧L is unsupported for this correlation and will be removed from the correlation table in the Post-correlation processing process. For the all-in-one correlation rule for A is A->(K∧L)∨G∨H, and thus only A-> K∧L, A->G, and A->H are supported correlation result.

In the correlation result generating process, the correlation table will be enclosed into correlation result message. In this step only the entry with its Guessed Number as 0 will be enclosed into correlation result. If the Guessed Number of all entries are not 0, then the whole correlation table will be enclosed into the correlation result message.

The generated correlation result message can be either echoed back into correlation message queue, or exported to higher level correlator, which depends on whether the symptom event of the correlation result/correlation table is in the export event set of this correlator.

After the Post-correlation, the whole correlation process for the assigned correlation task ends, and the correlation thread will then terminate.

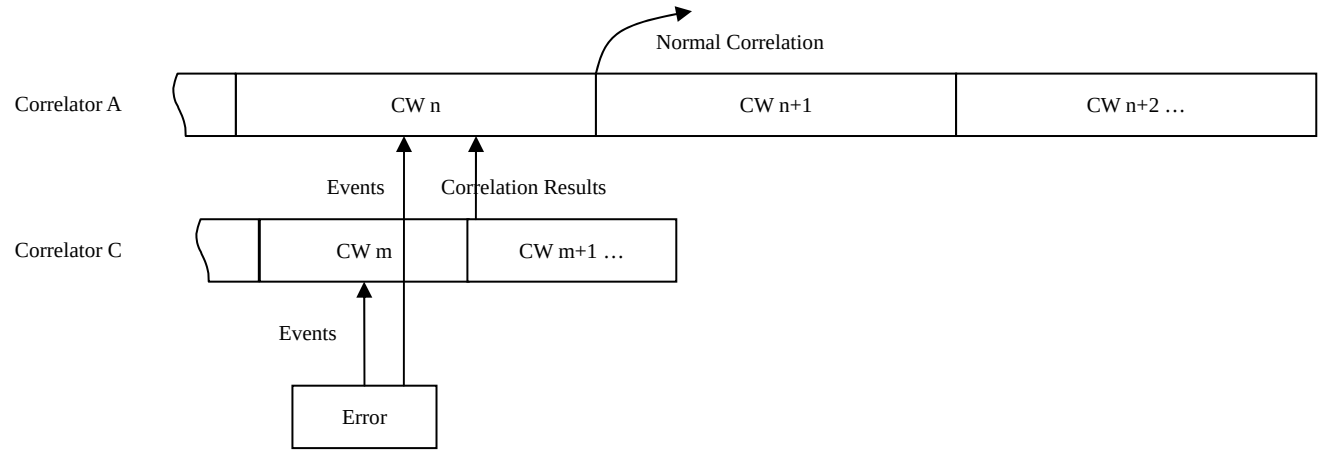
4.3.6 Delay correlation mechanism and delay window analysis

There are circumstances that the arrival of necessary correlation elements are sparse, which could cross two correlation windows. This could cause the frequent invocation of Event Guessing Process, which both increases the workload of the correlator (event missing correlation does much more work than normal correlation) and decreases the accuracy of correlation result. Therefore, some efforts should be made to reduce the frequent invocation of event guessing process.

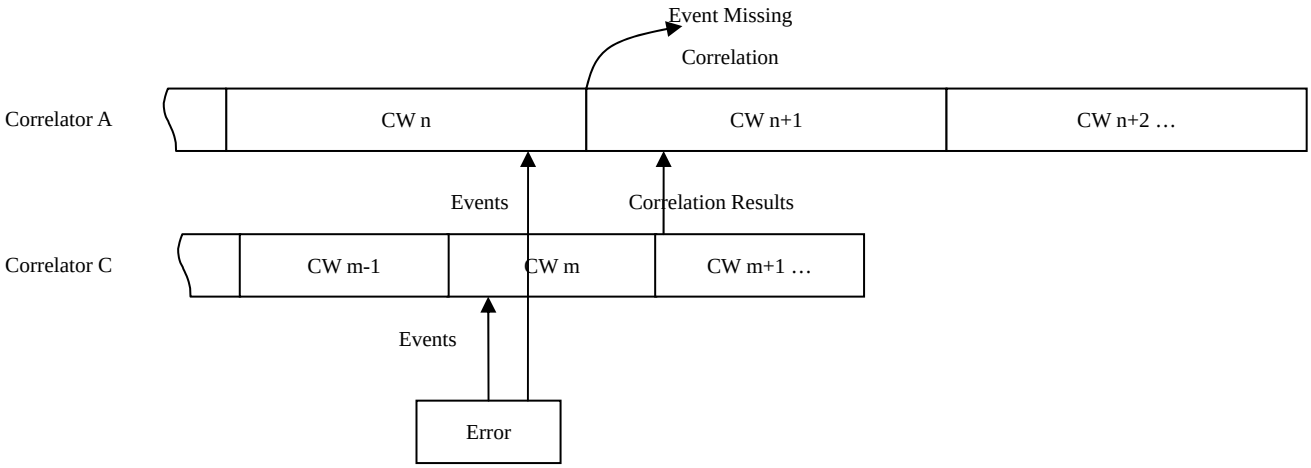
The circumstance of sparse arrival of correlation elements can be caused by two main reasons: network congestion and the difference of correlation windows among different correlators. The first reason is unpredictable and difficult to cater for, as it is caused by underlying network and is out of the control of the proposed fault system. However, the second reason is predictable and most sparse arrivals are caused by it. Therefore, a way should be found to reduce the frequency of invocation of Event Guessing process caused by this reason.

Delay Correlation is introduced to address this issue. Its main idea is to delay its correlation for a time period waiting for the un-received correlation elements. An example will be given here to explain the arrival pattern of correlation results.

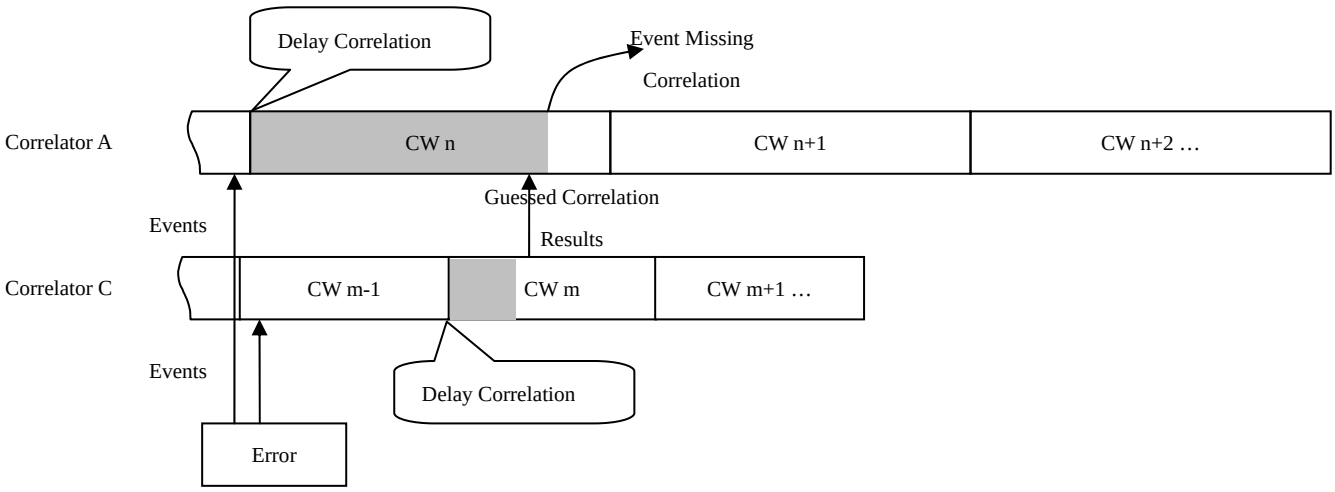
In most cases, as illustrated in Figure 4-17 (a), if an error occurred in managed network and was received during the correlation window m, and correlation window m will ends within the time scale of correlation window n, then both the symptom



(a) Normal Case



(b) Event Missing Correlation



(c) Worst Case

Figure 4-17 Event Arrival Patterns for event correlation

events triggered by the error and the correlation result calculated by Correlator C can be received within correlation window n. Correlator A can then calculate the root

cause of the received events without the invocation of event guessing process soon after correlation window n terminates.

However, there are circumstances that a low level correlation window crosses two adjacent high level correlation windows. As illustrated in Figure 4-17 (b), the correlation window m in correlator C cross both the correlation window n and correlation window n+1 in correlator A. Thus the symptom events caused by a network error can be received within correlation window n in correlator A. however, the correlation result that are necessary for the correlation n will not be available until correlation window n+1. Under this circumstance, unnecessary element guessing process will be called because of the delay arrival of low level correlation result.

This circumstance can be partially avoided by adopting delay correlation. If the delay correlation mechanism is adopted, when the normal correlation can not be performed, the correlation thread will first delay the correlation for a configurable time period waiting for the un-received correlation elements (both low level correlation results and delayed correlation events). If enough un-received correlation elements are received during the delay period, event guessing could be avoided.

There are currently no well-defined criteria for setting the length of a correlation window. However, the max-bound for the length of correlation window will be analyzed here in the remaining text. Figure 4-17 (c) gives the worst case under which the longest time should be delayed. In Figure 4-17 (c), the events for correlator A is received during the correlation window n-1, but the event for correlator C is received during the correlation window m-1. For the start times of correlation window m-1 is slightly later than the end time of correlation window n-1, according to the delay correlation mechanism given before, delay correlation will be performed. If very unfortunately, the elements received by correlator c in correlation window m-1 are not enough for normal correlation, so the correlator c will also perform delay correlation. Therefore, under the worst case, if the correlator is going to keep the invocation of

Event Guessing Process at the lowest level, the delay time for correlation A should be calculated like this:

delay(correlator A) = *cw*(correlator C) + *delay*(correlator C)

If we apply this to a more general circumstance, the longest delay time for correlator c (at the lowest possibility of invocation of Event Guessing Process) at correlator level i should be set to:

$delay_i(c)=max(cw_{i-1}(e)+delay_{i-1}(e)) \qquad e \in subBranch(c)$

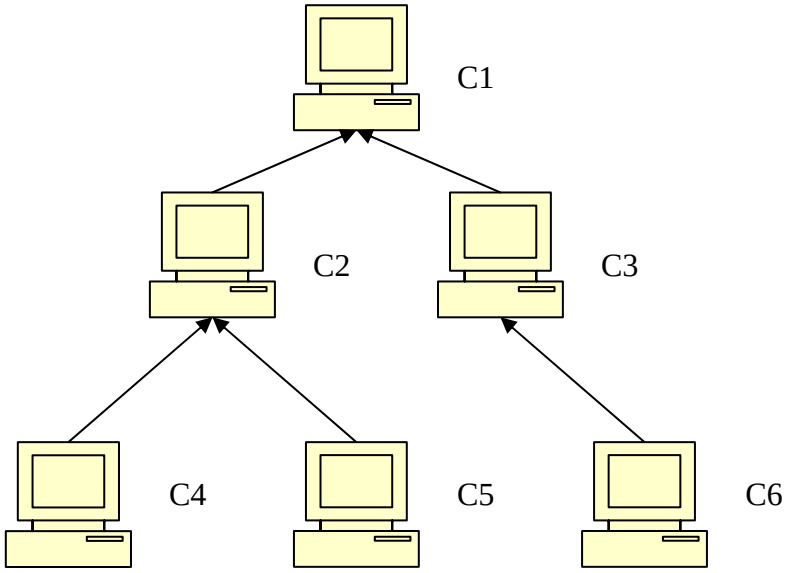


Figure 4-18 FMN topology

For example, in the FMN given in Figure 4-18 the max delay for C1 at level 2 should be:

$delay_2(C1)=\max((cw_1(C2)+delay_1(C2)), (cw_1(C3)+delay_1(C3)))$

Substituting $delay_{i-1}(e)$ by $\max(cw_{i-2}(f) + delay_{i-2}(f))$ $f \in subBranch(e)$ iteratively, we can calculate the delay for correlator c at level i should be:

$$delay_i(c) = \sum_{k=0}^{i-1} cw_k(e) + \max_{p \in subBranch(e)}(delay_0(p))$$ e is the correlator that holds the max delay at level k.

There is no low level correlator below level 1 correlator, so $delay_0(p)$ should be set to the *event branch radius* of branch of events in export event set of correlator p at level 0.

If we set the delay of a correlator to the max delay, the correlation precision will increase however the time consumed by delay maybe much longer than the time consumed by event guessing. Therefore, a tradeoff should be made between the correlation precision and correlation latency, and to select a property value for the correlator.

4.4 Correlation Expert Knowledge self-updating mechanism

Correlation Rules, Information on Correlation Event and Routing Ontology for KBN are three main expert knowledge types for event correlation. However the expert knowledge are network components (hardware, software and services) related. Therefore, the modification in any components will cause these expert knowledge types to go out of date, and thus will cause the newly modified component not to be managed by the proposed fault management system. In order to enable the proposed fault management system provide fault management for the newly modified network components, all related expert knowledge files, e.g. Correlation Rule, Routing Ontology, Correlation Event Information, and Fault Recovery Policies would need to be updated manually and re-compiled. Although the modification of components in the managed network may not be very frequent, the knowledge base updating would be a heavy and error-prone task.

In order to address this issue, a self-updating mechanism is proposed. The self-updating mechanism will be performed by adding network monitors such as connectivity monitor, service monitor and hardware monitor and so on as outside OSS

to the fault management system, and have them monitor different layers (hardware, software and service) of the managed network. Once there is a modification in the managed network, either caused by error or network operators' configuration, corresponding network monitor will notice it, and then send a modification message to the proposed fault management system.

In the proposed fault management system, several knowledge files as well as their access drivers that assist the updating of the aforementioned knowledge will be added, such as conceptual event correlation graph, components dependency graph, managed object taxonomy. Besides, an application module that holds the self-updating algorithm will be designed and developed. This module will work as a configuration manager in the proposed fault management system.

Currently this self-updating mechanism has not been well defined. Therefore, instead of a detailed design only high level verbal descriptions are available here. Thus the self-updating feature is listed as future work.

Chapter 5

Implementation

This chapter describes a partial implementation to the system described in chapter 4. The primary purpose of the implementation was to evaluate the distributed event correlation scheme that was designed. In the following subsections, the software and development environment, the implementation of the SNMP simulator, the implementation of event correlator and the implementation of knowledge base will be introduced.

5.1 Overview

The implementation of the proposed fault management system followed the following steps: implementation of the simulator, design of the format of expert knowledge file and implementation of knowledge base, and finally, implementation of the event correlator. An implementation level architecture of this system is given in Figure 5-1.

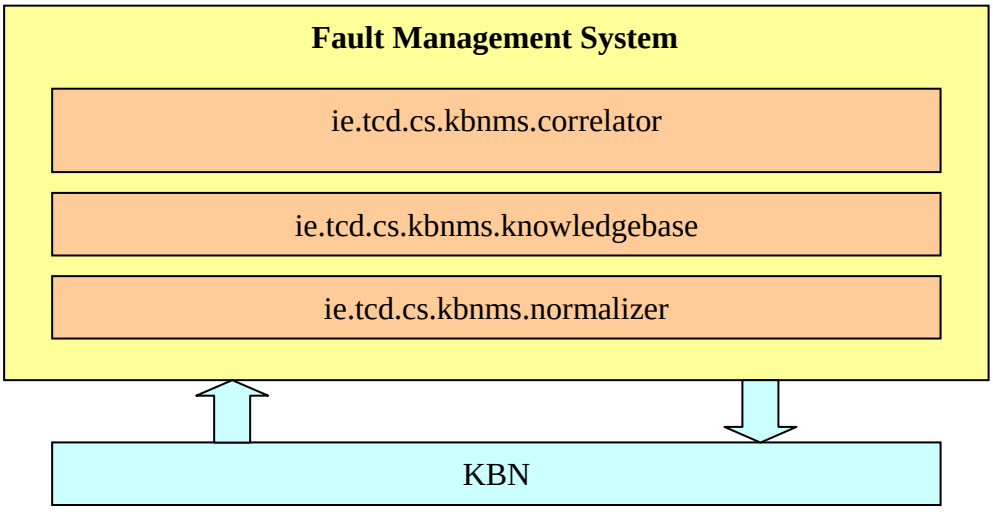


Figure 5-1 Implementation level architecture of this system

The implementation level architecture in Figure 5-1 is slightly different from the design given in Figure 4-2. The package ie.tcd.cs.kbnms.correlator is the implementation of the Service Organizer and Event Correlator service running on it; the package ie.tcd.cs.kbnms.knowledgebase is the implementation of the Knowledge Base; the package ie.tcd.cs.kbnms.normalizer is the implementation of the Event Normalizer. As this implementation is an evaluation version of the distributed correlator scheme, the Trouble Shooter has not been implemented, and for the sake of simplicity the Subscription Manager has been implemented by the class SubscriptionManager.class and has been put into the package ie.tcd.cs.kbnms.normalizer.

This evaluation implementation was Java based using: JDK version 1.5 build 11; the version of KBN used to support the running of the system is 3.0; and the IDE used for software development was Eclipse 3.2. All code, configuration files, and ontology files can be found in the accompanying CD.

5.2 Simulator Implementation

An event sender has been implemented to simulate the error messages (using SNMP traps) sent from the managed network. This simulator has been used in the evaluation to send synthetic SNMP traps wrapped by KBN notification. This simulator was developed based on SNMP4J 1.8.2 which is a java version SNMP implementation and extSiena which is the implementation of underlying KBN 3.0. Figure 5-2 illustrates the workflow of the event simulator (represented by ie.tcd.cs.taiw.extsnmp4j).

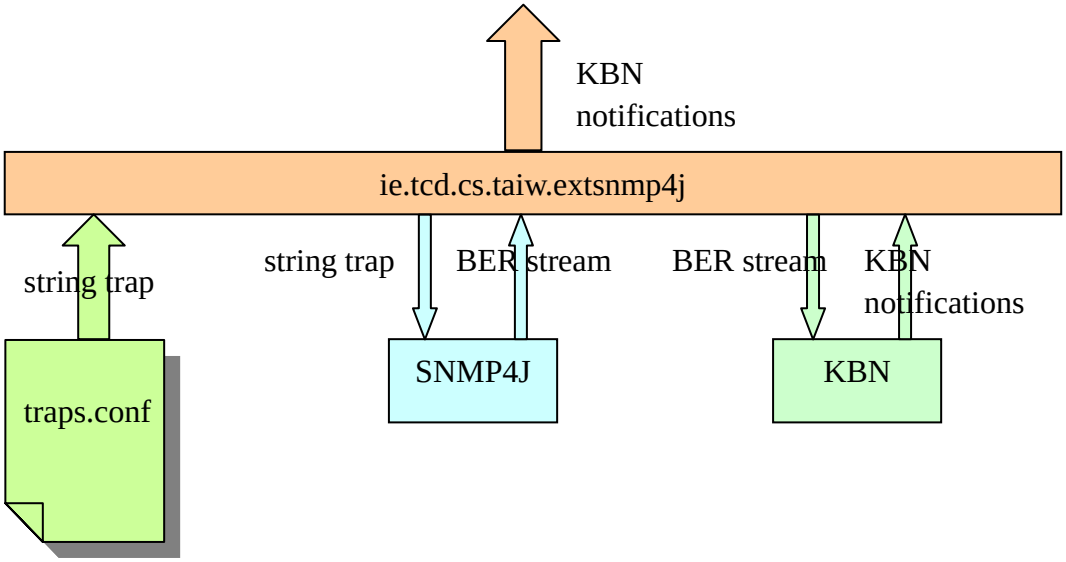


Figure 5-2 Implementation level architecture of event simulator

As illustrated in Figure 5-2, the whole work flow includes three steps:

(1) Reading synthetic SNMP field value from configuration file named traps.conf. Its format is given in Appendix B. Each SNMP field in traps.conf will be read according to its intrinsic data type, for example, the “community” field will be read as a string, the “error index” field will be read as Integer. Figure 5-3 gives a sample SNMP trap wrapped by a KBN notification.

Event=https://www.cs.tcd.ie/~taiw/ontology/2007/8/22/RoutingOnt.owl#C
MessageType=1
Sender="192.168.0.2"
SeqNo=1197308787787505792
Payload="\2474\002\004b\030\350'\002\001\000\002\001\0000&0\r\006\b+\006\001\002\001\001\003\000C\00170\025\006\n+\006\001\006\003\001\001\004\001\000\006\a+\006\001\002\201\024\001\000\000\000\000\000\000\000\000\000\000"

Figure 5-3 SNMP trap wrapped by KBN notification

(2) Constructing SNMP package and encoding the package into Basic Encoding Rule (BER) [21] stream. This step will be done by calling Pdu.encodeBER() in package org.snmp4j. The code snippet listing below illustrates the how a PDU is created.

```
/* Creating PDU */
PDU pdu = new PDU();
pdu.setType(pduType);
pdu.setRequestID(new Integer32(reqID));
pdu.setErrorStatus(errStatus);
pdu.setErrorIndex(errIdx);
pdu.addAll(vbs);
```

(3) Constructing a KBN notification and publishing it into the KBN. This step is done by constructing a ThinClient that connects to the KBN server, and then uses the publish() method to publish a notification. The code listing below gives an illustration as to how notifications are published.

```
/* creating ThinClient and publish notification */
ThinClient Client = null;
try{
    Client = new ThinClient("tcp:127.0.0.1:1234");
}catch(InvalidSenderException e){
    System.err.println("FrontendSimulator Error: can not create KBN \ Client.");
    return;
}
Client.publish(n);
```

5.3 Service Organizer Implementation

The package ie.tcd.cs.kbnms.correlator is the implementation of Service Organizer. For the sake of simplicity, the evaluation implementation incorporates both the Event Correlator service and Service Organizer into this package. Figure 5-4 gives the implementation level architecture of Service Organizer.

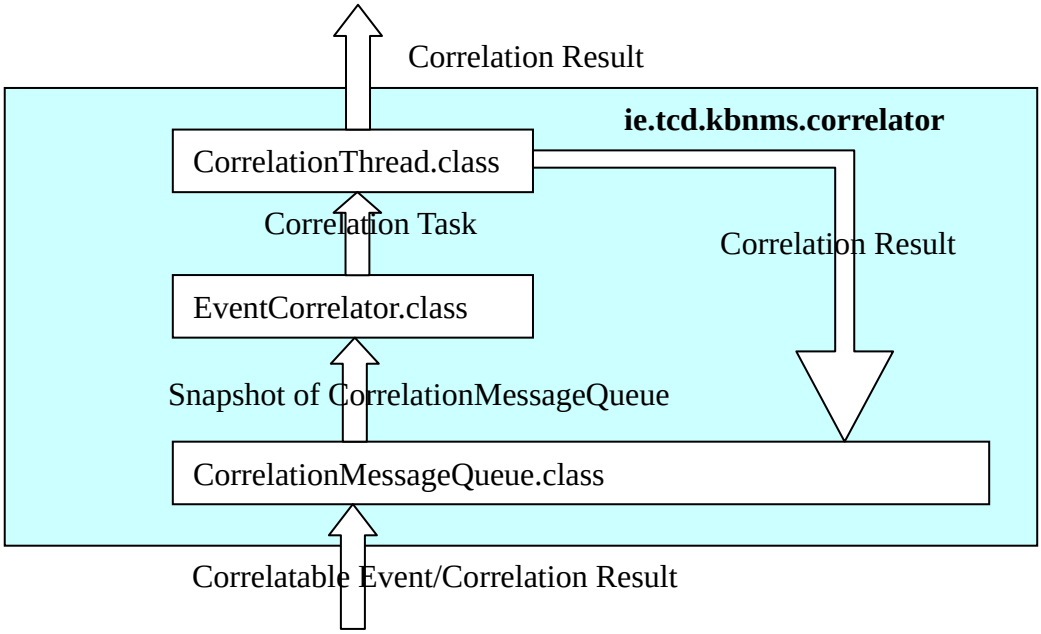


Figure 5-4 Implementation level architecture of event correlator

Class CorrelationMessageQueue is the implementation of Correlation Message Queue. In the evaluation implementation, it uses a java ArrayList to store all correlation elements it receives from lower layer components. All correlation elements (including correlation event and correlation result) inherit from one same interface Correlatable.class. This interface contains some common operation for correlation event and correlation result, such as getEvent() and getOccurrenceTime() and so on. In addition, inheriting from one same interface enables correlation events and correlation results can be kept in a single Correlation Message Queue.

The class EventCorrelator and class CorrelationThread implements the Event Correlator service. They work together and implement a multi-threaded event correlation service. As mentioned in section 4.3.4, the EventCorrelator will select correlation tasks from CorrelationMessageQueue after a correlation window ends and then assign the selected correlation tasks to idle CorrelationThreads. All correlation elements that are selected into correlation tasks will be removed from correlation message queue to keep the CorrelationMessageQueue thin. The code listing below illustrates the correlation task analysis process and the assignment of correlation tasks

to CorrelationThreads.

```
while(true){
    /* waiting for correlation window elapse */
    correlationWindowElapse();
    /* reset terminated correlation thread */
    resumeTerminatedThreads();
    if(CMQRef.isEmpty()){
        System.out.println("INFO (EventCorrelator): No correlatable message in \
Correlation Message Queue, enter another correlation round.\n");
        continue;
    }
    /* if the correlation message queue is not none, gain a snapshot of current
    correlation thread */
    CMQCopy = CMQRef.copy();
    /* perform correlation task analysis */
    getAllCorrelationBranches();
    if(TaskList.isEmpty()){
        /* no correlation tasks is gained */
        System.out.println("INFO (EventCorrelator): TaskList is null, enter another \
correlation round.");
        continue;
    }
    /* assign correlation task to correlation thread */
    generateCorrelationThread();
}
```

The Correlation for each correlation task will be performed in each CorrelationThread following the following 6 steps: (1) generating direct correlation rule; (2) initializing correlation table; (3) checking whether a delay correlation is needed; (4) merging correlation result into correlation table; (5) checking whether an event guessing

process is needed; (6) post-correlation process. The code listing below illustrates how these 6 steps are organized from a high point of view. For more detail on each step, please refer to code in class ie.tcd.cs.kbnms.correlator.CorrelationThread.

```
/* generating direct correlation rule */
DCR = getDirectCorrelationRule();
Vector<CorrelationTableEntry> ct = null;
while(Delayed == false){
    /* initialize correlation table */
    ct = initCorrelationTable();
    /* if delay correlation is needed, perform delay correlation */
    delayCorrelation(ct);
    if(HasNewElement == true){
        /* re-initialize correlation table given new elements are fetched */
        ct = initCorrelationTable();
    }
}
/* merging correlation result into correlation table */
Vector<CorrelationTableEntry> newCT =
populateCorrelationTableWithCorrelationResults(ct);
/* if event guessing process is needed, perform event guessing process */
if(needElementMissingCorrelation(newCT)){
    newCT = performElementMissingCorrelation(newCT);
}
/* post-correlation processing, including removing illegal potential correlation result,
shrink correlation table, generating correlation result and exporting correlation result
or echoing it back into correlation message queue */
removeUnsupportCorrelationTerm(newCT);
newCT = shrinkCorrelationTable(newCT);
CorrelationResult finalCR = genCorrelationResult(newCT);
procCorrelationResult(finalCR);
```

This package contains most of the intelligence of the proposed fault management system, and its development and debugging took the longest time in the whole development phase.

5.4 Knowledge Base Implementation

As this implementation is an evaluation version focused on the distributed correlation scheme, the implementation of the Knowledge Base mainly concentrates on the design and implementation of the format of expert knowledge such as event information, correlation rules and Event Correlation Graph and their access driver. Event information and Correlation Rules were encoded into the ontology named as tsecg.owl, and the Event Correlation Graph was encoded into the ontology named as RoutingOnt.owl. All these ontology files can be found in the accompanying CD.

The event information of an event mainly includes some relevant topology specific information of an event, for example, its sender, its UUID, its OID and so on. The ontology snippet listing below illustrates how the event information for the event A in Figure 4-10 is organized.

```
<Event rdf:ID="A">
    <sender rdf:datatype="&xsd;string">192.168.0.11</sender>
    <averageFaultDelay rdf:datatype="&xsd;int">30</averageFaultDelay>
    <isAbsoluteRootCause
rdf:datatype="&xsd;boolean">false</isAbsoluteRootCause>
    <multiSymptom rdf:datatype="&xsd;boolean">false</multiSymptom>
    <isObservable rdf:datatype="&xsd;boolean">true</isObservable>
    <displayString rdf:datatype="&xsd;string">A</displayString>
    <mapTo rdf:datatype="&xsd;string"
```

>https://www.cs.tcd.ie/~taiw/ontology/2007/8/22/RoutingOnt.owl#A</mapTo>

<oid rdf:datatype="&xsd;string"

>1.3.6.1.2.148.11</oid>

<uuid rdf:datatype="&xsd;string"

>0b34e3e8-aaaf-4f63-b1c9-fd9247a8a31f</uuid>

</Event>

The correlation rule is represented using two slots – cause and symptom – of an ontology class CorrelationRule. The ontology snippet listing below shows the ontology encoding of correlation rule A->C∧D.

<CorrelationRule rdf:ID="CorrelationRule_40">

<cause rdf:resource="#C"/>

<cause rdf:resource="#D"/>

<symptom rdf:resource="#A"/>

</CorrelationRule>

The access driver for tsecg.owl is implemented by class TopologySpecificEventCorrelationGraphImpl.class. This class implements the interface TopologySpecificEventCorrelationGraph which works as the common access interface for Topology Specific Event Correlation Graph. This driver is implemented using Jena, which is a java framework for building semantic web application.

In this implementation, Correlation Rule is represented by a class called CorrelationEquation.class. The “and or” format of cause and symptom are represented the combination of two supportive classes that are designed to represent the disjunction relationship (Disjunction.class) and conjunction relationship (Conjunction.class). The calculation rules for correlation rules are represented by two methods (*merge*() and *substituteBy*()) in class CorrelationEquation. Besides, the absorbtion rule and duplication rule from propositional logic are also introduced into

this implementation to support correlation rule calculation.

The routing ontology (in RoutingOnt.owl) is constructed using OWL subclass/superclass relationship. In this implementation it is mainly used by KBN nodes to route trap messages in KBN and it is also used by Subscription Manager to perform subscription management. It is constructed after the example given in Figure 4-10. The ontology snippet from routing ontology listing bellow gives an example how events are organized in this ontology. It lists all events that as the “subclass” of event C.

```
<owl:Class rdf:ID="C">
    <rdfs:subClassOf rdf:resource="#A"/>
</owl:Class>
<owl:Class rdf:ID="G">
    <rdfs:subClassOf rdf:resource="#C"/>
    <rdfs:subClassOf rdf:resource="#D"/>
</owl:Class>
<owl:Class rdf:ID="K">
    <rdfs:subClassOf rdf:resource="#C"/>
</owl:Class>
```

5.5 Event Normalizer Implementation

The normalizer implementation includes the implementation of Outgoing Message Queue, Event Dispatcher and Message Processing Model. For the sake of simplicity, the evaluation implementation did not implement the Incoming Message Queue, and besides, the implementation of subscription manager (SubscriptionManager.class) has also been incorporated into the event normalizer.

The Outgoing Message Queue and Event Dispatcher are implemented by class file

OutgoingMessageQueue.class and MessageProcessor.class separately. The design level Message Processing Model is implemented by two interfaces and one abstract class; they are MessageProcessingModel, MessageParser and Message (abstract class). The abstract class Message represents the message object. All message types that need to be supported by the proposed fault management system should inherit from this class. The interface MessageParser is mainly in charge of encoding/decoding messages between network stream and message object. The MessageProcessingModel contains two methods: prepareOutgoingMessage() and processIncomingMessage(). The prepareOutgoingMessage() is in charge of compiling KBN notifications from message object. The processIncomingMessage(), however, will do more than just turn a notification into message object. It has the ability to understand the content of the received message and then perform corresponding operations according to the content of received message. The code snippet list bellow illustrates how the incoming SNMPTrapMessage is processed by processIncomingMessage() in SNMPTrapMessageProcessingModel.

```
/* parsing notification and construct message object */
CorrelatableMessage msg = (CorrelatableMessage)MP.parse(n);
Correlatable crrMsg = msg.asCorrelatable();
if(crrMsg == null){
    return;
}
/* perform corresponding operation to the message */
CorrelationMessageQueue.getInstance().add(crrMsg);
```

In the evaluation implementation, the SNMP trap message will only be used for correlation, so currently instead of performing content analysis the processIncomingMessage() of SNMPTrapMessageProcessingModel simply put the received SNMPTrapMessage into correlation message queue.

In this evaluation implementation only two Message Processing Models are

implemented to support the normal running of distributed event correlation scheme; they are SNMP Message Processing Model and Correlation Result Message Processing Mode. Figure 5-5 introduces the architecture of event normalizer at implementation level.

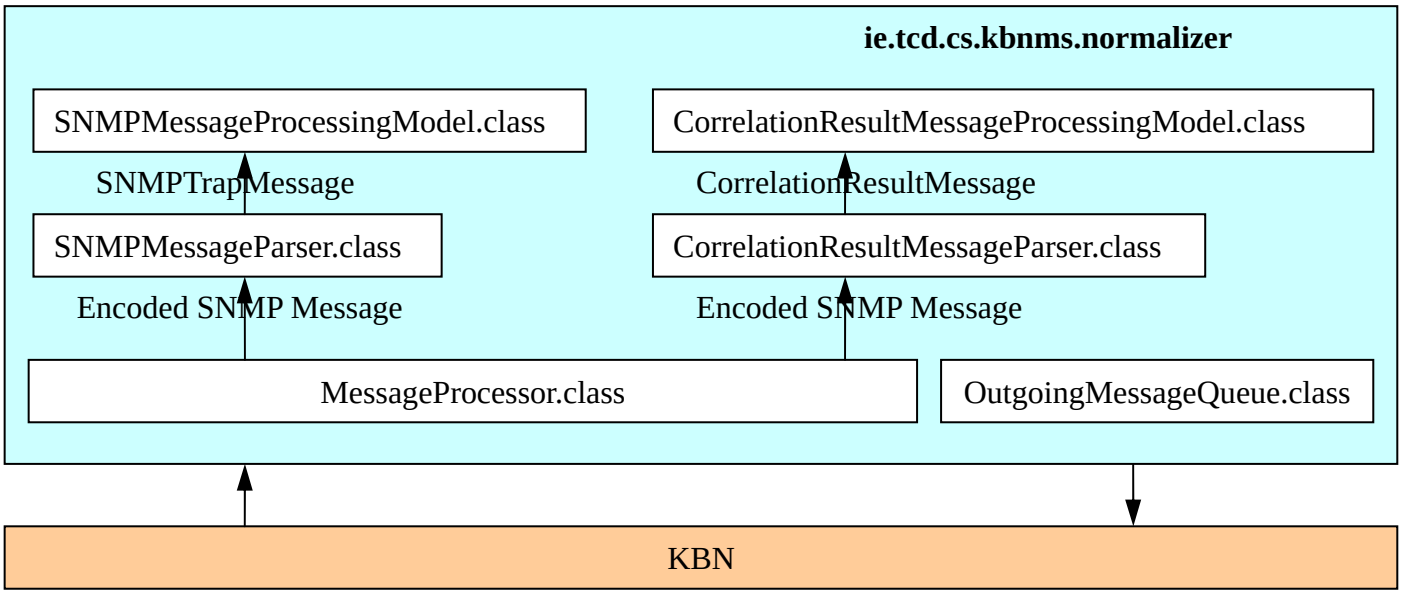


Figure 5-5 Implementation Level architecture of Event Normalizer

5.6 Implementation Issues

This implementation version is mainly developed for the evaluation of the proposed correlation scheme. Therefore, issues encountered during the implementation mainly reside in the developing and debugging of the distributed correlation scheme.

(1) There were always concurrent access exceptions thrown when debugging the multi-threaded event correlation scheme. For simplicity to address this, I simply set all methods in correlation message queue and its queue entry using synchronized keyword. Although easy this method however degrades the performance of the code. In the future this critical section will be carefully analyzed and reimplemented to decrease the granularity of object lock.

(2) The circumstance that two correlation tasks could the same correlation event was

not noticed at the beginning of the development, which assumes only one correlation task has the “shared” event. I address this problem by using a counter to indicate how many tasks will use this event, and then divide the correlation task analysis into two steps: scan step and event fetching step. The counter will be added by twice in the scan step if two tasks will use this event, and in the event fetching the event will be duplicated to every correlation task before it being removed.

(3) Besides the above mentioned two issues during the implementation period, the usage of new tools such as Jena and snmp4j required significant effort, especially as these were new to me.

Chapter 6

Evaluation

The Evaluation was conducted in two parts: performance benchmark and feature comparison. In the performance benchmark part, test cases were set up and carried out under different architectures and configurations in order to gather the performance statistics of the proposed correlation scheme. Analysis was then undertaken on these performance statistics in order to get its performance feature. The feature comparison part was conducted from two perspectives: the architecture and the correlation scheme. The architecture comparison compares the functional features between systems with different architecture, e.g. centralized architecture and peer-to-peer architecture; the correlation scheme comparison compare the proposed correlation scheme with current existing several correlation algorithms such as rule-based scheme, codebook scheme and Artificial Intelligence based scheme from the functional point of view.

6.1 Performance Benchmark

A Performance Benchmark was designed to test the timing performance of the proposed event correlation scheme. A correlation graph with 13 events and a maximum correlation depth of 4 was constructed. Test cases were constructed in accordance with this correlation graph. A test case set included 13 test cases which tested the performance of either normal correlation or event missing correlation at different correlation depth was constructed. The correlation scheme was written in Java and was compiled by jdk1.5.0_11. The benchmark environment is listed in table 6-1, and all test cases is listed in appendix A.

CPU:	Intel Celeron^{m} 1.7GHz
Memory:	768MByte
Operating System:	Microsoft Windows XP SP2
Java Environment:	JDK1.5.0 build 11
KBN Version:	3.0

Table 6-1 Benchmark environment.

6.1.1 Benchmark Criteria, Methodology and Results

Benchmark metrics for the proposed correlation scheme include Average Correlation Time, Average Branch Selection Time, Direct Correlation Rule Calculation Time, Correlation Table Calculation Time, and Post-correlation Processing Time. The Average Branch Selection Time is the time that the correlator selects correlation tasks from correlation message queue, and assigns them to correlation threads. The Direct Correlation Rule Calculation Time is the time that a correlation thread spends on calculate direct correlation rule, the correlation table calculation time is the time that a correlation thread spends on calculating correlation table and the Post-correlation Processing Time is the time that the correlation thread spends on removing illegal correlation table entry, shrinking correlation table and pushing correlation result into KBN. The Average Correlation Time is the total time that a correlation thread takes to perform the correlation for a task. It is the sum of Direct Correlation Rule Calculation Time, Correlation Table Calculation Time and Post-correlation Processing Time.

These evaluation criteria are selected mainly for the reason that they can represent the correlation speed of the proposed correlation scheme. Furthermore, each the selected evaluation criterion represent a critical step in the whole correlation phase. Thus by observing the percentage of a specific correlation step occupies in the whole correlation, we can have a clear picture of which step needs further improvement in order to improve the performance of the whole correlation.

The performance test was conducted under two architectures – single centralized correlator and multiple distributed correlators – in order to gain the performance statistics of the proposed correlation scheme when applied to different architecture. All test cases were carried out more than 20 times. For each time, all the aforementioned benchmark metrics were measured. After the 20 rounds execution for one test case, an arithmetic mean value for each benchmark metric was calculated. All test cases in the test case set were carried out in the performance test on single server, and because of the feature of distributed correlation, only several test cases from the test case set are selected to conduct performance test over multiple correlator. Table 6-2 lists the result of performance test on single correlator and Table 6-3 lists the result of performance test over multiple correlators. All metrics in Table 6-2 and Table 6-3 are represented using millisecond and because both KBN node and correlators are running on the same machine and sharing a single CPU time in this simulation, the statistics gained from this performance benchmark will be slightly larger than those from real-world use.

Test Case Name	Number of Thread	Average Branch Selection Time	Average Correlation Time	DCR Calculation Time	Correlation Table Calculation Time	Post-correlation Processing Time
Norm_2_2	1	10.5	23	18.74	1.03	4.64
Norm_2_3	1	11.0	26.05	19.95	1.85	5.42
Norm_2_4	Thread-1	25	64.55	40.05	1.32	23.52
	Thread-2		32.55	18.55	1.07	13.04
Norm_3_3_a	1	25.05	120.14	94.64	1.85	25.49
Norm_3_3_b	1	27.1	79.15	54	1.42	23.41
Norm_3_4_a	1	25.5	65.04	44.09	1.23	24.46
Norm_3_4_b	Thread-1	46.67	217.69	179.23	3.82	48.25
	Thread-2		123.85	76.92	1.53	44.31

Norm_3_5	1	30.53	127	100	1.46	30.42
Norm_3_7	Thread-1	48.24	210.59	177.06	5.32	40.33
	Thread-2		114.71	71.76	1.94	43.68
Norm_4_5	1	48.39	322.21	278.65	1.76	43.24
Norm_4_13	1	50.42	325.33	283.25	2.13	40
EMC_2_2	1	10.2	1031.59	19.41	1.25	11.03
EMC_2_3	1	13.53	1025.76	18.75	2.11	9.41

Table 6-2 Test case execution result over 1 correlator

Test Case Name	Number of Correlator/Thread	Average Branch Selection Time	Average Correlation Time	DCR Calculation Time	Correlation Table Calculation Time	Post-correlation Processing Time
Norm_3_5	2/2	33.00	113.00	86.00	3.2	22.00
Norm_3_5	2/3	31.00	73.19	48.81	1.88	28.75
Norm_4_13	3/4	51.67	59.33	51.75	1.67	51.75

Table 6-3 Test case execution result over multiple correlators

6.1.2 Statistics Analysis

- Average Branch Selection Time

The major factors that affect the Average Branch Selection Time are the number of correlation branches that exist in correlation message queue, and the number of event in each selected correlation branch, as well as the depth of the selected correlation branch. By examining Table 6-2 we can know that the unit Average Branch Selection Time is approximate 11ms/branch for a correlation depth of 2, 25ms/branch for a correlation depth of 3 and 48ms/branch for a correlation depth of 4. In addition, the test cases with two correlation branches have their Average Branch Selection Time value two times the value of unit Average Branch Selection Time of the same correlation depth, e.g. Norm_2_4 has its Average Branch Selection Time as 25ms, which is two times as the unit Average Branch Selection Time (11ms), and it is the

same with Norm_3_4_b and Norm_3_7. The Event Missing Correlation put little affects on Average Branch Selection Time, and therefore the EMC_2_2 and EMC_2_3 hold the same Average Branch Selection Time with Norm_2_2 and Norm_2_3.

By comparing the corresponding test case in Table 6-2 and Table 6-3, we can know that when applied to multiple correlators, the Average Branch Selection Time changes little, so the distribution of correlation puts little affects on the Average Branch Selection Time.

A conclusion can be made from the previous analysis that the major cause that affects the Average Branch Selection is the depth of correlation and the branch number in a correlation round. When applied on a single correlator, the increase of Average Branch Selection Time with depth follows exponential incremental model of power 2, and the increase of Average Branch Selection Time with branch number follows linear incremental model (Figure 6-1).

$$ABST(dpth, brch) = ABST(dpth - 1, brch) * 2$$
$$ABST(dpth, brch) = ABST(dpth, brch - 1) + unitABST(dpth)$$

Figure 6-1 Average Branch Selection Time incremental model

- Average Correlation Time, DCR Calculation Time, Correlation Table Calculation Time and Post-correlation processing time

The Average Correlation Time is the sum of DCR Calculation Time, Correlation Table Calculation Time and Post-correlation processing time. As indicated by Table 6-2, when applied to a single correlator, the Correlation Table Calculation Times for all test cases vary little with the correlation branch; the length of Post-correlation Processing Time relate mainly to the length of correlation result, and it also varies little with the correlation depth. The DCR calculation time, however, occupies most of the Average Correlation Time, and it varies significantly with the number of event

existing in the correlation branch and correlation depth. Therefore the DCR calculation time is the major cause that affects the length of the Average Correlation Time when the proposed correlation scheme is applied on single correlator.

However, when applied on multiple correlators, there is a significantly performance improvement in the proposed correlation scheme. By comparing the corresponding executing result in Table 6-2 and Table 6-3, we can know that all the test case result improves significantly. The performance improvements are listed in Table 6-4.

Test Case Name	Number of Correlator/Thread	DCR Calculation Time Decrease (%)	Average Correlation Time Decrease (%)
Norm_3_5	2/2	14.00%	11.02%
Norm_3_5	2/3	51.19%	42.37%
Norm_4_13	3/4	81.73%	81.76%

Table 6-4 Performance Improvement

All test cases have significant performance improvement when carried out over multiple correlators. The more correlators and threads this correlation is distributed, the higher performance it will improve. According to the proposed correlation scheme, multiple servers can distributed the DCR calculation process and parallel the correlation by having low level correlation finished within the correlation window of high level correlation, and then both simplify the DCR calculation process for high level correlation and save the correlation time for high level correlation. Therefore the overall time for normal correlation can be greatly improved and the delay put by underlying infrastructure will then become the major cause that affect the correlation time and correlation accuracy.

One thing worth mentioning is that the Average Correlation Time of Element Missing Correlation takes more than 1000 milliseconds and much larger than that of normal

correlation. This is because the lost event triggered the delay correlation, and the delay time is set to 1000 milliseconds for all test cases. After removing the delay time from the Average Correlation Time for Element Missing Correlation, the statistics for Element Missing Correlation are nearly the same as normal correlation for the proposed correlation scheme processing the normal correlation and element missing correlation using the same approach.

Pressure and accuracy testing will be conducted in the future to test the maximum events number that this proposed correlation scheme can afford at a given correlation speed and accuracy.

6.2 Feature Comparison

The proposed fault management system is a pure distributed fault management system. When speaking about the pure distributed fault management, it means that not only the architecture of the organizational structure is distributed, but also the correlation scheme is distributed and thus has the ability to distribute a single correlation task over multiple event correlators. In the following text, comparisons between the proposed fault management scheme and other existing fault management schemes from both the architecture point of view and correlation scheme point of view will be given.

6.2.1 Architectural Comparison

In the architecture comparison section, two projects with two typical architectures – OSI network management model and Madeira project – are selected to conduct the comparison. OSI network management model use the manager/agent architecture with most management intelligence residing in manager. This architecture can reduce the complexity and difficulty for the deployment of network management system. When applied to small scale network, the centralized architecture may have better

performance than distributed architecture, because the distributed architecture will put a lot delay on the transmission of management information between management servers. However, because of the centralized architecture, the manager has to process all the management messages from its agents. Therefore the centralized manager puts a performance bottleneck in the whole system and makes the whole system less robust facing the event storm. The proposed fault management system uses a pure distributed architecture, thus it has better event correlation abilities and higher event receive abilities. These make the proposed fault management much faster than centralized network management model in root cause reasoning and more robust in dealing with the event storm.

The Madeira project, however, adopts the hybrid peer-to-peer architecture, that is to say, it combines the hierarchical architecture with peer-to-peer architecture. This can provide better performance in event correlation and can provide a high throughput for the network event. The Madeira project was directly constructed on traditional network transport protocol and the current development of publish/subscribe system could put a negative affect on the throughput of the proposed fault management system. Therefore, the throughput of the proposed fault management system could be less than this project. However, the intrinsic feature of publish/subscribe system – only the subscribe notification can be forwarded to the subscriber – can protected the management server from been attacked by malicious codes or hackers, which can not be provided by traditional network. Thus the proposed fault management system can provide a higher security level than Madeira project.

It is clear that the centralized management model is more suitable for the management of small scale network, but it is difficult to deal with the management of large scale network because of its centralized architecture and low throughput. The proposed fault management system could be lower in event throughput than distributed network management system constructed directly on traditional network, but it is higher in security, and with the development of publish/subscribe technology and hardware the

throughput could be promoted to an acceptable level.

6.2.2 Correlation Scheme Comparison

In this section, a comparison between the proposed event correlation scheme with currently existing event correlation schemes is provided. Traditional rule based correlation, code book based correlation and artificial intelligence based correlation will be selected to perform the comparison.

Traditional rule based correlation scheme will match the incoming event with rules until the matching reaches a static state. Therefore, a large amount of expert knowledge is required to construct the knowledge base and correlation rule. Besides, the incoming event should be accurate. Any lost or delay in the event will cause the failure of correlation. The proposed correlation scheme, however, provide a scheme to automatically populate the knowledge base and correlation rules, and its Delay Correlation approach and Element Missing Correlation approach make the correlation available even under the circumstance that events are delayed or even lost.

Codebook approach is similar to traditional rule based approach, but it is much higher in correlation speed and is more robust in facing with the delay and lost of events, because of it uses minimum hamming distance to determine the correlation result. The correlation speed of codebook approach may have faster correlation speed than the proposed correlation approach when applied to one correlator. However, the codebook approach can not be distributed and has less flexibility in expanding its correlation ability. The proposed correlation approach has more flexibility in performance expansion. It can have its correlation speed and event throughput increased by simply plugging in new correlator and sharing the correlation task. Besides, another drawback existing in Codebook approach is that it can not handling temporal event. The proposed correlation scheme handle temporal event by adding in a temporal event monitor. Once the incoming event satisfies the given temporal policies, a temporal

event will be issued.

The Artificial Intelligence correlation scheme is a correlation scheme that radically different from rule based approach and codebook approach. It mainly uses the Bayesian Network and causal graph to perform the correlation, and it is currently still under research. Most AI correlation scheme is still centralized approach, so they still low in event throughput when applied to large scale network. However, the AI approach is a promising approach applied to network management. Its self-learning ability can greatly reduce the time of Delay Correlation and increase the accuracy of Element Missing Correlation. Therefore, it is the future work of this project to add in AI technologies to perform correlation.

6.3 Evaluation Conclusion

From both the performance benchmark point of view and the feature comparison point of view, the proposed fault management approach has both drawbacks and merits. It expresses an exponential increase of the Average Branch Selection Time with the correlation depth increases. This is obviously not scalable and will affect the max correlation depth in a single FMS. Besides, for the proposed correlation scheme perform correlation by iteratively calculating correlation table, its correlation speed could be lower than codebook approach (using Boolean operator compare only once) *in single correlator*. However this correlation scheme uses adjustable weighted belief factor to determine the correlation result, which has more flexibility and accuracy in determine the correlation result than just using hamming distance. Furthermore, when distributed among multiple FMS, there will be a dramatic performance promotion in the correlation speed. Its distributed architecture, high correlation speed on multiple correlators, high security level, high throughput, supporting for Element Missing Correlation, automatically knowledge base modification ability, and the capability to allow easy plug in of correlators makes it a promising approach.

Chapter 7

Issues

Issues and future work of the proposed fault management system mainly reside in the distributed correlation scheme. For example, currently there are no well defined criteria for the setting of the length of correlation window for a given correlator. The length of correlation window for a correlator will greatly affect both the speed and precision of a correlation. A too big correlation window could increase the correlation precision but will increase the response time for an error, and a too small correlation window will improve the response time for an error but will decrease correlation precision. The criteria for finding a tradeoff between the response time and correlation precision have not been set up yet, and for current implementation, the setting of correlation window is manually undertaken by the system administrator. A self-learning process could be added into the proposed system to find the best length for correlation window for correlator.

The second issue is the setting of the length that a correlation event/ correlation result can stay in correlation message queue. The adoption of a correlation window will mean some “too late” received correlation event/ correlation result will be dead (will not be used by other correlation) in the correlation message queue, which means they will not be used by other correlation anymore. Furthermore, the echoed correlation result may not be used by other correlations. According to the current design, the message collector will run periodically to recycle those dead correlation events; and the trouble shooter will also run periodically to pick the out-of-date correlation results to perform fault recovery process. However, how long is the best time to regard a

correlation event/ correlation result as dead is an issue that exists in the current proposed system. If the length for this is too long, the correlation message queue will become too long and thus affect the performance of event correlator; on the other hand, the correlation result/ correlation event may be recycled or used by trouble shooter before they could be used by other correlation. Criteria for setting the “dead length” should be set up to find a best trade off between the length of correlation message queue and the hit rate.

There is no well-defined criteria for selecting the belief factors for correlation result and their weight, and therefore the currently used belief factors (Guessed Number, Lost Number, Mis-matching) and weight can not perfectly represent the belief degree of a correlation result. Thus the identification of criteria for selecting the belief factors and their weight is required urgently.

The current correlator controls the part of the Event Correlation Graph on which its correlation is going to perform by putting subscriptions to the underlying KBN node. Therefore, if the Event Correlation Graph that the correlator is going to perform correlation on is too fragile, a great number of subscriptions will be put to KBN in order to precisely describe its interests, which will greatly increase the comparison time performed by KBN and thus increase the delay put by KBN. Currently there is not a good solution to address this issue.

Chapter 8

Conclusion and Future work

8.1 Achievements

This research partially achieves its original goal. A new architecture of a fault management system that runs on a semantic publish/subscribe system is proposed. In addition, a pure distributed correlation scheme with high fault tolerant capability has been proposed. A partial implementation to the proposed fault management system has been finished, and a simulator has also been implemented.

8.2 Highlights

There are two highlight in this research. First, this research incorporates the semantic publish/subscribe mechanism into a fault management system, which shifts the addressing mechanism of events to the underlying infrastructure. Thus no attention is needed to be paid to the addressing problem by both the managed network and fault management system. Furthermore, the event filtering mechanism of KBN can increase the robustness of the fault management system to a certain extent when in the face of event storms.

The distributed correlation scheme is another highlight of this research. This correlation scheme is a pure distributed correlation scheme which distributes a single correlation task over a whole correlator network. This can have one correlation running parallel on several correlators and increase the correlation performance.

Besides, the correlation behaviour is controlled by subscription and thus the correlation behaviour of a correlator can be simply changed by changing the subscription to a different part of event correlation graph. This increases the flexibility of the system and enables the enhancement of performance of this fault management system through adding in new correlators.

8.3 Future work

Further work could be done to finish the design and implementation of the self-updating mechanism proposed in section 4.4. This enables the proposed fault management system to have the ability to be informed when changes have been made in the management network and then perform changes to the corresponding expert knowledge in the knowledge base. In order to enable the self-updating mechanism of the proposed fault management system, a new self-updating algorithm as well as its related expert knowledge bases should be developed. In addition, support from other OSS systems, such as service monitor, connection monitor or hardware monitor and so on should also be provided.

The PnP feature proposed in section 4.2 will also be another candidate for future work of the proposed fault management system. By the addition of a PnP feature, the correlator can be added into a correlator network without too much explicit configuration; and in addition, other correlators can be made aware of the leaving of a correlator, either because of fatal failure or being removed by system administrator, and thus the job of the departed correlator will be soon taken by other correlators. The PnP feature can decrease the dependency of the whole fault management system to specific correlators, and thus makes the system more autonomous in handing the fatal fault that may arise in the fault management system.

The addition of AI technologies such as belief network into the correlation algorithm

is also a meaningful future work to this project. The probabilistic theory can be used to calculate the belief degree while performing event guessing process; this can greatly improve the hit-rate of guessing correlation. The self-learning mechanism can also be used in this system to partially address the timing issue introduced in chapter 7. For example, the length of correlation window, delay window can then be set dynamically through the self-learning ability.

Besides the aforementioned future work, unfinished aspects of the design should be completed, for example, the architecture design of Front End; the design and implementation of the Trouble Shooter, the fault recovery policies file, and the fault recovery access driver.

[1] Rouvellou, I.; Hart, G.W., "Automatic alarm correlation for fault identification," *INFOCOM '95. Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People. Proceedings. IEEE* , vol., no., pp.553-561 vol.2, 2-6 Apr 1995

[2] Bouloutas, A.T.; Calo, S.; Finkel, A., "Alarm correlation and fault identification in communication networks," *Communications, IEEE Transactions on* , vol.42, no.234, pp.523-533, Feb/Mar/Apr 1994

[3] Jaesung Choi; Myungwhan Choi; Sang-Hyuk Lee, "An alarm correlation and fault identification scheme based on OSI managed object classes," *Communications, 1999. ICC '99. 1999 IEEE International Conference on* , vol.3, no., pp.1547-1551 vol.3, 1999

[4] Hasan, M.; Sugla, B.; Viswanathan, R., "A conceptual framework for network management event correlation and filtering systems," *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on* , vol., no., pp.233-246, 1999

[5] Yu, M., Li, W., and Chung, L. W. 2006. “A practical scheme for MPLS fault monitoring and alarm correlation in backbone networks”. *Comput. Networks* 50, 16 (Nov. 2006), 3024-3042. DOI= http://dx.doi.org/10.1016/j.comnet.2005.11.006

[6] Yemini, S.A.; Kliger, S.; Mozes, E.; Yemini, Y.; Ohsie, D., "High speed and robust event correlation," *Communications Magazine, IEEE* , vol.34, no.5, pp.82-90, May 1996

[7] Chi-Chun Lo; Shing-Hong Chen; Bon-Yeh Lin, "Coding-based schemes for fault identification in communication networks," *Military Communications Conference*

Proceedings, 1999. MILCOM 1999. IEEE , vol.2, no., pp.915-919 vol.2, 1999

[8] B. Gruschke. “Integrated event management: Event correlation using dependency graphs”. In *Proc. IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, Oct. 1998.

[9] D.W. Guerer, I. Khan, R. Ogler, R. Keffer, An Artificial Intelligence Approach to Network Fault Management, SRI International, 1996.

[10] Dilmar Malheiros Meira, A Model For Alarm Correlation in Telecommunications Networks, Federal University of Minas Gerais, PhD Thesis, Nov. 1997.

[11] Hewlett Packard, HP OpenView event correlation for the telecommunications environment: Technology brief, Sep. 1995.

[12] M. Moller, S. Tretter, and B. Fink. Intelligence filtering in network management system. In IFIP/IEEE International Symposium on Integrated Network Management, IV (ISINM’95) [1995], page 304-315.

[13] Simona Brugnoni, Guido Bruno, Roberto Manione, Enrico Montariolo, Elio Paschetta, and Luisella Sisto. An expert system for real time fault diagnosis of the Italian telecommunications network. In IFIP International Symposium on Integrated Network Management, III (ISINM’93) [1993], pages 617-628.

[14] Yemini, Y., "The OSI network management model," *Communications Magazine, IEEE* , vol.31, no.5, pp.20-29, May 1993

[15] Katchabaw, M. J., Lutfiyya, H. L., Marshall, A. D., and Bauer, M. A. 1996. Policy-driven fault management in distributed systems. In *Proceedings of the the Seventh international Symposium on Software Reliability Engineering (ISSRE '96)*

(October 30 - November 02, 1996). ISSRE. IEEE Computer Society, Washington, DC, 236.

[16] ITU-T, M.3000 Overview of TMN Recommendations, International Telecommunication Union, Feb. 2000.

[17] ITU-T, M.3010 Principles for a telecommunications management network, International Telecommunication Union, Feb. 2000.

[18] Pablo Arozarena, Martijn Frints, Sandra Collins, Liam Fallon, Martin Zach, Joan Serrat, and Johan Nielsen. Madeira: A peer-to-peer approach to network management, Wireless World Research Forum, Sep. 2006

[19] Zach, M.; Parker, D.; Nielsen, J.; Fahy, C.; Carroll, R.; Lehtihet, E.; Georgalas, N.; Marin, R.; Serrat, J., "Towards a framework for network management applications based on peer-to-peer paradigms The CELTIC project Madeira," *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP* , vol., no., pp. 1-4, 2006

[20] E. Rosen, Y. Rekhter, RFC 4364 BGP/MPLS IP Virtual Private Networks (VPNs), IETF RFC, Feb. 2006.

[21] Telecommunication Standardization Sector of ITU, X.690 Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), International Telecommunication Union, Jul. 2002.

[22] Michael K. Smith, Chris Welty, Deborah L. McGuinness. OWL Web Ontology Language Guide, In W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-guide-20040210/, Feb. 2004.

[23] Brian McBride, RDF Primer, In W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. Feb 2004.

[24] Daconta, M., Obrst, L., & Smith, K. (2003). The Semantic Web. Indianapolis: Wiley Publishing.

[25] T. Berners-Lee, R. Fielding, L. Masinter. RFC 2396 Uniform Resource Identifier (URI): Generic Syntax, IETF RFC, Aug. 1998.

[26] Graham Klyne, Jeremy J. Carroll, Brian McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax, In W3C Recommendation, http://www.w3.org/TR/rdf-concepts/, Feb. 2004.

[27] Dave Beckett, Brian McBride, RDF/XML Syntax Specification, In W3C Recommendation, http://www.w3.org/TR/rdf-syntax-grammar/, Feb. 2004.

[28] Masahiro Hori, Jerome Euzenat, Peter F. Patel-Schneider, OWL Web Ontology Language XML Presentation Syntax, In W3C Recommendation, http://www.w3.org/TR/owl-xmlsyntax/, Jun. 2003.

[29] Dan Brickley, R.V. Guha, Brian McBride, RDF Vocabulary Description Language 1.0: RDF Schema, In W3C Recommendation, http://www.w3.org/TR/rdf-schema/#ch_summary, Feb. 2004.

[30] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Perter F. Patel-Schneider, Lynn Andrea Stein, OWL Web Ontology Language Reference, In W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-ref-20040210/, Feb. 2004.

[31] A. Carzaniga, D. S. Rosenblum, and A. L.Wolf. Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems, 19(3), Aug. 2001.

[32] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In Proceedings of ICDCS, Jun. 1999.

[33] Segall, B., Arnold, D., Boot, J., Henderson, M., Phelps, T., Content-Based Routing with Elvin4, In Proceedings AUUG2K, Canberra 2000.

[34] Keeney, J.; Lewis, D.; O'Sullivan, D.; Roelens, A.; Wade, V.; Boran, A.; Richardson, R.; Runtime Semantic Interoperability for Gathering Ontology-based Network Context; Network Operations and Management Symposium, 2006. 10th IEEE/IFIP 2006 Page(s):56 – 65

[35] J. Case, M. Fedor, M. Schoffstall, J. Davin, RFC 1157 A Simple Network Management Protocol (SNMP), IETF RFC, May 1990.

[36] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, RFC 1901 Introduction to Community-based SNMPv2, IETF RFC, Jan. 1996.

[37] K. McCloghrie, D. Perkins, J. Schoenwaelder, RCF 2578 Structure of Management Information Version 2 (SMIv2), IETF RFC, Apr. 1999.

[38] D. Levi, P. Meyer, B. Stewart, RFC 3413 Simple Network Management Protocol (SNMP) Applications, IETF RFC, Dec. 2002.

[39] R. Presuhn, RFC 3416 Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP), IETF RFC, Dec. 2002.

[40] Keeney, J., Lewis, D., O'Sullivan, D., "Ontological Semantics for Distributing Contextual Knowledge in Highly Distributed Autonomic Systems", Journal of Network and System Management, Special Issue on Autonomic Pervasive and Context-aware Systems, Volume 15, Number 1 / March, 2007. doi:10.1007/s10922-006-9054-5

[41] R. Presuhn, RFC 3418 Management Information Base (MIB) for the Simple Network Management Protocol (SNMP), IETF RFC, Dec. 2002.

[42] R. Raghunarayan, RFC 4022 Management Information Base for the Transmission Control Protocol (TCP), IETF RFC, Mar. 2005.

[43] E. Rosen, A. Viswanathan, R. Callon, RFC 3031 Multiprotocol Label Switching Architecture, IETF RFC, Jan. 2001.

[44] Chuck Semeria, RFC 2547bis: BGP/MPLS VPN Fundamentals, Juniper Networks, Inc. White Paper, Part Number: 200012-001 03/01. Jan. 2003.

[45] D. Allan, T. Nadeau, RFC 4378 A Framework for Multi-Protocol Label Switching (MPLS) Operations and Management (OAM), IETF RFC, Feb. 2006.

[46] Y. Rekhter, T. Li, S. Hares, RFC 4271 A Boarder Gateway Protocol 4 (BGP-4), IETF RFC, Jan. 2006.

[47] J. F. Huard. Probabilistic reasoning for fault management on XUNET. Technical Report, Center for Telecommunications Research, Columbia University, New York, NY, 1994

Appendix A

Design of Test Cases

Name	Correlation Depth	Number of Event	Number of Branch
Norm_2_2	2	2	1
Norm_2_3	2	3	1
Norm_2_4	2	4	2
Norm_3_3_a	3	3	1
Norm_3_3_b	3	3	1
Norm_3_4_a	3	4	1
Norm_3_4_b	3	4	2
Norm_3_5	3	5	1
Norm_3_7	3	7	2
Norm_4_5	4	5	1
Norm_4_13	4	13	1
EMC_2_1	2	1	1
EMC_2_2	2	2	2

Note: Norm stands for Normal Correlation. EMC stands for Element Missing Correlation

Table A-1 Design of Test Cases

Appendix B

Format of trap simulator configuration file

The configuration file for trap simulator is used to hold synthetic SNMP traps, and simulator will construct SNMP traps using synthetic SNMP traps in this configuration file. A configuration file can hold more than one SNMP traps. Figure B-1 gives the format of a SNMP trap.

```
{{}{(String)sender=taiw;}{tcp:127.0.0.1:1234;}{v2c}{public}{}{0}{0}{SNMP
v2-MIB:sysUpTime=123456789}{SNMPv2-MIB:snmpTrapOID=NET-SNMP-
EXAMPLES-MIB:netSnmpExampleHeartbeatNotification}{NET-SNMP-EXA
MPLES-MIB:netSnmpExampleHeartbeatRate=1000}}
```

Figure B-1 Synthetic SNMP trap

Each term in the a synthetic SNMP trap is parenthesized by a pair of curly bracket. The first term is left for the arrangement of the sending pattern, e.g. one SNMP trap for every 1 second and so on. Currently, the sending pattern has not been implemented yet, and it will be left blank, which means this trap will be only sent once. The second term ({(String)sender=taiw}) is the term for configuring the KBN header. Its format follows (type)tag_name=value. The current KBN implementation only provides support for types such as string, byte array, long, Boolean and bag. Terms from the third to the last are arranged according to the sequence of fields of SNMP. Currently, these terms are arranged according to the sequence of fields in SNMPv2c.