Reduction of TCO by Efficient Development of ICT Operation Systems and Intelligent Fault Analysis

Submitted to
Graduate School of Information Science and Technology
Osaka University
January 2009

Toshio TONOUCHI

List of Publications

Journal Papers

1. Toshio TONOUCHI, and Shin NAKAJIMA, “Implementation of a Fast Q3 Agent Platform for Agents Embedded in Network Elements”, IPSJ Transactions Vol.41 No.4 pp.1226-1233, May. 2000. (in Japanese)
2. Toshio TONOUCHI, and Shin NAKAJIMA, “A Toolkit for Developing DSL Translator”, IPSJ Transactions Vol.43 No.1 pp.146-155, Jan. 2002. (in Japanese)
3. Toshio TONOUCHI and Masayuki MURATA, “Root cause analysis technique for derivative failures with implicit dependencies”, accepted in IEICE Transactions on Communications, Vol.J92-B, No. 8, Aug. 2009. (in Japanese)

Refereed Conference Papers

1. Toshio TONOUCHI, Takashi FUKUSHIMA, Asuka MANKI, and Shin NAKAJIM, “An Implementation of OSI Management Q3 Agent Platform for Subscriber Networks”, in Proceedings of IEEE International Conference on Communication (ICC), pp. 889-893, 1997

Non-Refereed Technical Papers

1. Toshio TONOUCHI, Masahiro TAKEI, Shin NAKAJIMA, and Shouichiro NAKAI, “A Memory Management Architecture of MIB for OSI System Management”, IEICE Society Conference No.2 p.116, Sep. 1995. (In Japanese)
2. Toshio TONOUCHI and Shin NAKAJIMA, “A Main-memory MIB Platform for Fast and Compact OSI Management Agents”, The 52th National Convention of IPSJ No.1 pp.99-100, Mar. 1996. (In Japanese)
3. Toshio TONOUCHI and Shin NAKAJIMA, “Architecture of OSI MIB Platform and its Performance Evaluation”, JSSST 13th Annual Conference, Sep. 1996. (In Japanese)
4. Toshio TONOUCHI and Shin NAKAJIMA, “A Re-targetable GDMO Translator”, JSSST 14th Annual Conference pp. 1-4, Sep. 1997. (In Japanese)

5. Toshio TONOUCHI and Shin NAKAJIMA, “A Template Driven method for Developing DSL Translators and Implementation of a Translator Toolkit based on the Method”, JSSST 16th Annual Conference, pp.189-192, Sep. 1999. (In Japanese)
6. Toshio TONOUCHI, “A Mathematical approach to definition and fulfillment of requirements for development of a policy management tool”, Technical Report of IPSJ (2001-SE-135-1) Vol.2001, No.114 pp. 1-8, 2001/11/21
7. Toshio TONOUCHI, Class of Service with an Access-control Policy System, IPSJ SE Object-oriented Symposium (OO), pp.50-58, 2002. (In Japanese)
8. Toshio TONOUCHI, Tomohiro IGAKURA, Naoto MAEDA, and Yoshiaki KIRIHA, “Policy Transition Mechanism: A New Approach to Multi-mode Management”, in Proceedings of IFIP/IEEE Network Operations & Management Symposium (NOMS) Application Session, pp.191-204, Apr. 2004

Preface

Information and Communication Technology (ICT) Systems have become indispensable elements in social infrastructures. High availability of the systems is seriously required. There have already happened a lot of cases where failures, such as malfunction and performance degradation, made large impacts on the society. Therefore, the management and maintenance of the ICT systems is essential efforts to keep their service level, to avoid failures, and to recover immediately when failures occur.

However, the management and maintenance costs of the ICT systems occupy a lot of portion of the Total Cost of Ownership (TCO) of ICT systems. In addition, the more functional and more complex ICT systems become, the more sophisticated management of the ICT systems is required. Therefore, the management and maintenance costs are increasing.

Management and maintenance costs include the following two:

- Capital investment cost of operation systems, and.
- Operation cost for daily maintenance.

Firstly, in this thesis, we propose an efficient management agent platform, which can efficiently run even on an inexpensive and poor environment. We also propose a method with a Domain Specific Language (DSL) approach for developing an agent application on the platform. These two are ones of solutions for the reduction of the capital investment cost of operation systems.

Secondly, we propose a fault analysis method of reducing the operation cost of the ICT systems. Administrators always watch whether managed systems run correctly or not. If failures happen on the systems, they have to analyze the cause of the failures as soon as possible. The detection requires a lot of knowledge of the managed systems for the developers, and it is a tough job. We propose a fault analysis method can reduces a burden of the administrators, and it results in the reduction of personnel costs used in the operation. We mention three contributions of this thesis in the following.

Firstly, we developed a Q3 management agent platform embedded in network elements.

The standardized specifications of the Q3 agents have rich functionalities. They, therefore, require a lot of hardware resources for the Q3 agents, such as much capacity of memory. Management agents embedded in network elements placed in access networks is so many that cost reduction of the management agents is important. We revise the structure of the containment tree and other data structures stored in Q3 agents, and we successfully run a Q3 agent on a PowerPC board with small capacity of memory.

Secondly, the capital investment cost of management agents includes the development cost of a Q3 agent application program as well as the cost of hardware where the Q3 agent runs. We propose a DSL approach in order to reduce the development cost. We point out one of the important classifications of maintenance, which is defined as an affected maintenance. It was about 8% of total maintenances in an existing project. In order to reduce the cost of the affected maintenance, we develop a DSL toolkit called Rosetta. DSLs developed with Rosetta can be easily maintained. We show that Rosetta can reduce the affected maintenance cost more than the visitor design pattern can.

The Q3 agent platform and Q3 agent application programs developed with Rosetta are actually used in network elements in base stations of intercontinental telecommunication cables.

Finally, we propose a fault analysis method for reducing the operation cost in the fault management. The proposed method can analyze a root cause from a lot of events issued by derivative failures. There are many works of fault analysis methods for managed systems with fixed structure. However, there appear a lot of distributed systems in these days. The configurations of the distributed systems are flexible, and the relations of failure derivation also are changeable. In addition, failure derivation occurred in the application program of the managed systems is implicit to the administrators, and it is difficult to find the root cause of derivative failures. Our method learns a failure derivation model from an existing event log, and it analyzes a root cause of the managed systems in run time by using the model. We show, in our experiments, that the method can correctly learn the failure derivation model, and the proposed method is effective if the parameters of the method are suitable to administrators' policy.

Acknowledgements

First of all, I would like to express my sincere gratitude to Professor Masayuki Murata of Graduate School of Information Science and Technology, Osaka University, under whose direction the studies described in this thesis have been carried out. I also thank to Professor Koso Murakami, Professor Makoto Imase, and Professor Teruo Higashino of Graduate School of Information Science and Technology, Osaka University, and Professor Hirotaka Nakano of Cyber Media Center, Osaka University, for reviewing this thesis.

We also thank Professor Shin Nakajima of National Institute of Informatics, who was my boss when he worked in NEC Corporation. He gave important advices to my work especially in a Q3 agent platform and its development methods. I appreciate a lot of helps of Mr. Yoshiaki Kiriha, who is also my ex-boss. Mr. Koichi Konishi, who is my boss, helps that my job in my business hour goes well. It is a great help for my schoolwork in private time goes well.

I would express my thanks to Professor Satoru Kawai of the Open University of Japan. He was my counselor in my master course of University of Tokyo. He gave comments on my work after I graduated from the master course.

I deeply thank my wife and my children. They are always encouragements to me. Especially my wife, Noriko, willingly admitted my trial to the doctor degree. I spent a lot of private time to the schoolwork, and it was a burden to my wife. I give thank to my wife for her a lot of efforts. I also thank my parents. They usually took care of my children when my wife and I were busy. They are also great encouragements to me.

Contents

List of Figures

List of Tables

Chapter 1
Introduction: Reducing TCO by reduction of management costs

Chapter 1. Introduction: Reducing TCO by reduction of management costs

1-1 Background

Information and Communication Technology (ICT) Systems have become indispensable elements in social infrastructures. High availability of the systems is seriously required. As they are getting more important, there happen a lot of cases where failures, such as malfunction and performance degradation, had large impacts on the society. Table 14 shows some accidents in recent years. Therefore, the management and maintenance of ICT systems is indispensable efforts to keep their service level, to avoid failures, and to recover them from failures immediately when the failures occur.

However, the management and maintenance costs of the ICT systems occupy a lot of portion of the Total Cost of Ownership (TCO) of ICT systems. Figure 1 shows the cost structure of ICT systems. This shows that the maintenance cost is almost equal to the system development cost. Figure 2 shows the ratios of investments in new ICT projects and in existing ICT projects. The ratio of the investment in existing ICT projects has been getting increasing from 2003 to 2007. In addition, the more functional and more complex the ICT systems become, the more sophisticated management of the ICT systems is required. Therefore, the management and maintenance costs are increasing. If the maintenance cost can be reduced, the large part of TCO costs will be reduced. Moreover, the reduction of the management and maintenance costs may contribute to increase of investments to new ICT projects and it may turn into a new ICT services. Because the business environments are changing so rapidly, the investment to new ICT systems are strongly required.

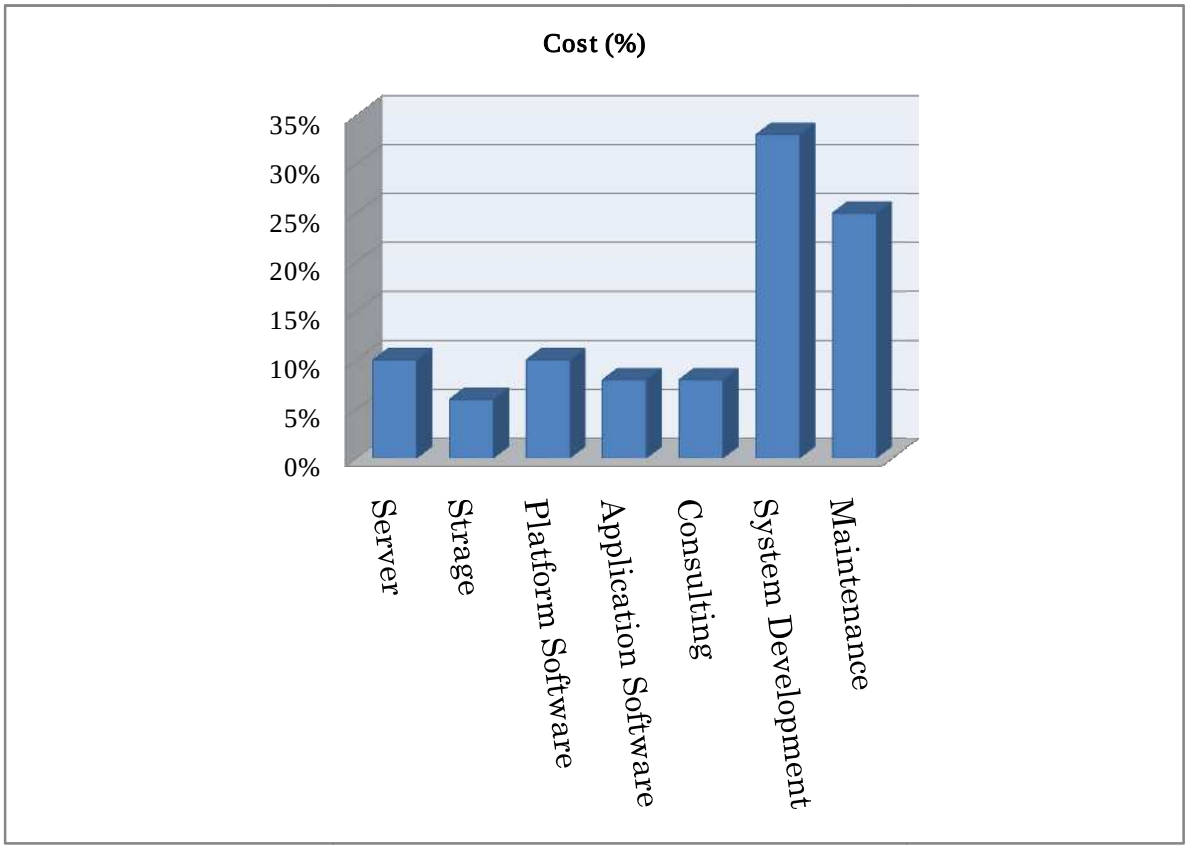

Figure 1: Cost structure of ICT systems [Gartner]

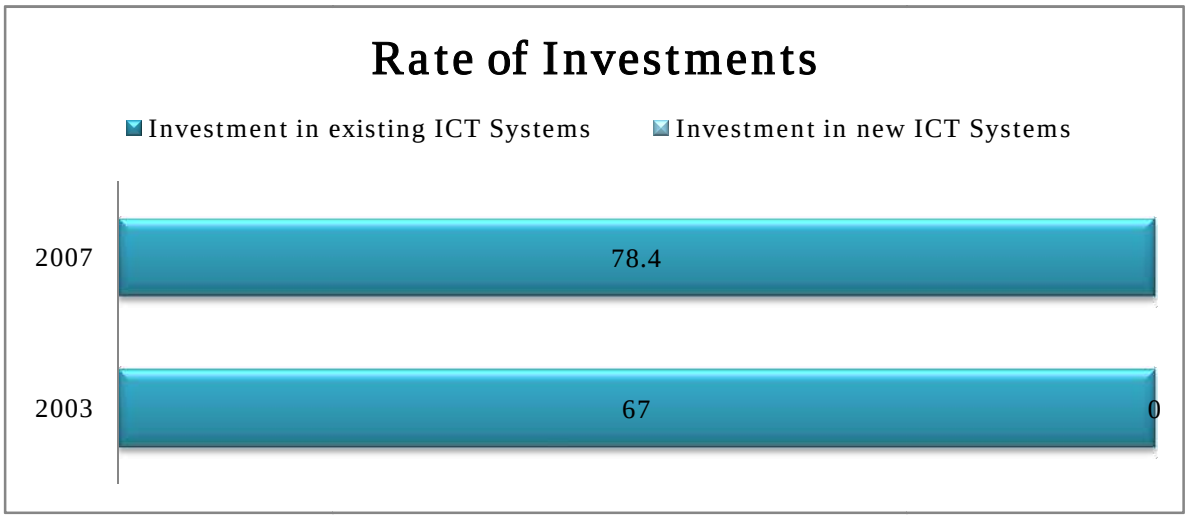

Figure 2 : Total Cost of Ownership in Japan [Gartner]

Chapter 1. Introduction: Reducing TCO by reduction of management costs

Management and operation costs include the following two:

- Capital investment cost of operation systems, and.
- Operation cost for daily maintenance.

The former cost consists of hardware cost necessary to the operation systems and the development cost of new operation systems. This is shown in Figure 3.

In order to reduce capital investment cost of operation systems, this thesis argues the two view of points: one is a view of architecture of operation systems, and the other is a view of development method of operation systems.

The personnel cost for the administrators is a large part of the operation cost. In order to reduce the personnel costs of daily operations, we focus on fault management area in this thesis. It is said that network management operations consist of FCAPS: Fault management, Configuration management, Account management, Performance management, and Security management. Watching whether systems are normal or abnormal is daily operations and is an important operation in order to keep the availability of the ICT systems. Fault managements are important but cost-consuming operation because, in the fault management, it takes times to identify where a fault happens and what kind of the fault is. That is reason why we struggle with the fault management. We propose a fault analysis method, which can reduce the burden of the administrators. It turns into the reduction of the personnel costs of fault management. It also contributes to rapid fault detection and, as a result, rapid fault recovery.

These three cost elements are closely related. For example, if the management agent can get more precise information, the fault analysis in the operation will become easier. However, such agent requires much memory resources in which the precise information are stored. It turns into high cost. We should think these three elements totally, but, in this thesis, we think them independently.

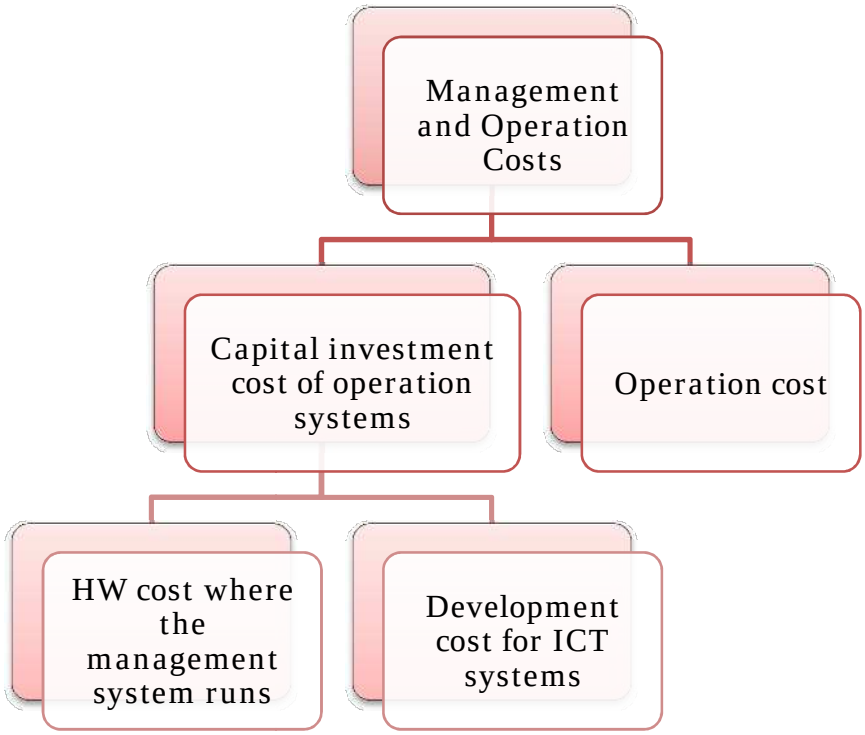

Figure 3: Cost structure of management and operation cost

1-2 Outline of this thesis

As shown in Section 1-1, the reduction of maintenance cost of ICT systems is a big issue. Firstly, in Chapter 2, we review the history of management and identify the trends so far. We propose three approaches to solve the issue. Firstly, we propose a network operation system platform, which enable a rich function network management agent in low cost in Chapter 3. Secondly, we also propose a development method of operation systems in Chapter 4. It can reduce the development costs of network management agents. Finally, we introduce a fault analysis method in Chapter 5.

Chapter 1. Introduction: Reducing TCO by reduction of management costs

1-2-1 High cost-performance Network Operation Platform [1][2][3][4][5]

Higher management functions were required in telecom network than SNMP [6] in IP networks. For example, high-availability and assured quality have been required in telecom networks, while IP networks are based on best-effort communications. Q3 management interface including Common Management Information Protocol (CMIP) [7] is standardized in ITU-T. CMIP has rich functions, such as a flexible naming structure called a containment tree. However, operation systems with the Q3 interface are too complex for efficient implementation. It was said that Q3 requires a high capability of necessary hardware and, as a result, expensive machines. An operation system comprises of a manager system and a lot of management agents, as shown in Figure 4. A management agent is often embedded in a network element (NE), it monitors the NE, and the information gotten from the NE by the agent is sent to the manager system. The manager system shows the information to human administrators. An agent system is required to be at low-cost because many agents are embedded in a lot of network elements. The cost of an agent largely impacts on the total cost of whole network.

We have planned to implement a robust agent system. Management agents must be robust because the agents are widely deployed. In other word, a failure in a management agent requires much cost. We choose a one-board computer system with main memory disk. Main memory disks are more robust than hard disk systems.

In addition, because data stored in main memory disk can be accessed much faster than that stored in hard disk drive, operation systems with man memory disk have a potential to achieve efficient operations of a complex Q3 interface. However, the cost of main memory is larger than that of hard disk.

In our approach, we deeply examined requirements of Q3 specifications, and we found that three kinds of memory-effective data structure satisfy the requirements. As a result, we can achieve both low cost and effective software platform, on which the effective agent can be implemented.

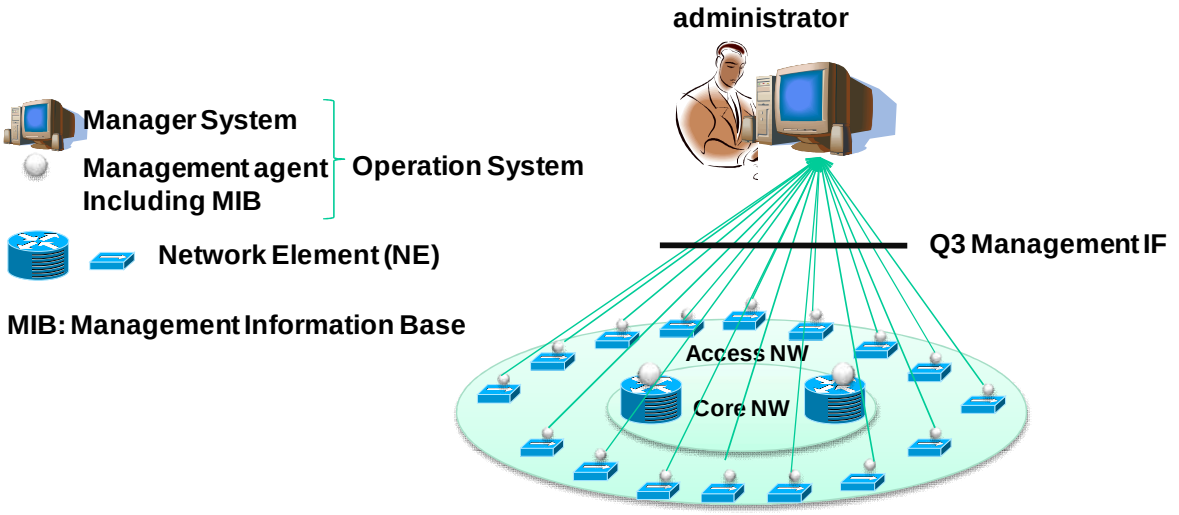

Figure 4: Operation System

1-2-2 Reducing development costs for Operation Systems [8][9][10][11]

We argue, in Section 1-2-1, the efficient agent platform running on an inexpensive machine environment. However, the cost of an agent system includes development cost of a Q3 application program running on the platform as well as the hardware cost. A Q3 agent system must be tailored to each kind of network elements because managed information is different from NEs. The development cost of a Q3 agent application for each NE must be low, and the agent application must be implemented as soon as possible. In Q3 agent system, each information model of each NE is defined by using GDMO templates [12]. We developed a GDMO translator which generates C++ program codes from the GDMO templates. The program codes generated from the GDMO translator are assumed to run on the agent platform mentioned in Section 1-2-1. It can decrease the development cost.

However, we have another problem, which is related to maintenance cost. Consider that the agent platform may be updated. For example, when a new platform for a new CPU or new OS is released, existing program codes cannot run on the new platform. In that case, the GDMO translator is also re-developed for a new platform in order to generate the program codes running on the new platform. It results in high maintenance costs. We propose a flexible Domain Specific Language (DSL) toolkit, which can easily modify how program codes the translator generates. We also developed a GDMO translator by using this DSL toolkit. This can reduce the maintenance costs of the GDMO translator and, as a

result, that of Q3 agent systems.

1-2-3 Low cost management for High-dependable Systems [13][14][15]

Management operation is a kind of a routine work. An administrator always watches the situation of a managed system, and sees whether an anomalous situation occurs or not. In usual cases, no anomaly occurs, but in a few cases, anomaly occurs. In the anomalous case, the administrator tries to find out the cause of failures and to fix it. He also escalates the trouble ticket to experts of the systems when he cannot find the cause. Therefore, in anomalous case, many human resources are consumed. It takes a lot of personnel costs, and it also decreases the availability of the managed system.

Some mission critical systems, such as telecom operator's systems, include a fault tolerant mechanism, but the administrator, of course, has to identify the cause of failure even after the stand-by systems work.

Several factors make it difficult to find the cause. One of them is failure derivation. A fault may cause several failures and error events. It is difficult for the administrators to find the cause from a lot of error events. Some kinds of faults have an explicit failure derivation relation. For example, a fault in a radio access controller (RNC) in radio access network derives failures in its subordinate node-Bs.

In addition, failure derivations among software components are not clear because the inside of software is opaque and because the relation of the failure derivations among them is complex.

We propose a machine learning method, in which failure derivation relation is a-priori learned from existing event logs. The proposed method can generate a failure derivation model, and it uses the model in order to find the fault, in other word a root failure, from a lot of events.

Chapter 2
History of ICT managements

We look back over the history of ICT managements in Figure 5. There are four trends.

- Remote operation
- Expansion of management area.
- Improvement of cost-performance of operation systems.
- Automation of management

We show each trend shown above.

2-1 Remote operation

In 1960 – 80, an administrator sent from a computer vendor always stays in his customer side and watches a main frame computer. In case of trouble, he rapidly fixes it. However, trend of downsizing of computer hardware in 1990's increase the ratio of the cost for the administrator relatively. In addition, the trend of distributed computing makes it difficult for the administrator to manage widely distributed network elements of managed systems. Therefore, it is required that the administrator can watch, through network, NEs distributed. Management protocol SNMP [6] was standardized by IETF in 1987. Management information accessed with the SNMP was also standardized, such as MIB-II [16]. The standardization means that common parts of NE can be managed even if the vendor of NE is different and that managed systems consist of NEs of several vendors can be managed. Several management protocols and information models are proposed for its management domain. Some of them are introduced in Section2-2.

2-2 Expansion of management area.

The targets of SNMP are network elements, such as a router. This is called element management. The scope of management has been extending from element management to network management, service management, and business management. eTOM [17], which is a map of management operations, was standardized. It includes whole the operations of telecom operators.

Another extension is an expansion of target domain. The target of SNMP is an IP network while that of CMIP is a telecom network. The target of WBEM/CIM [18] is an enterprise network including computer systems as well as routers. The target of TR069 [19] is management of home gateways.

Recent years, management area is including ubiquitous network, whose concept

includes sensors and actuators indirectly related to the physical world [20][21][22][23][24][25][26].

2-3 Improvement of cost-performance of operation systems.

As the target of management operations extended, as mentioned in Section 2-2, the rich functions are also required. For example, management protocol CMIP and model-definition language GDMO are standardized for management for telecom operators. These standardizations include a lot of rich management functions in order to achieve high reliable communication services. For example, they include high extensibility in order to model many kinds of network elements. This flexibility enables telecom operators to afford new-coming elements. Object-oriented concepts have been adopted for the extensibility. This extensibility increases the development cost of operation systems because the operation systems must be tailored to each kind of network elements. Researches for reducing the development cost were done. TNMS Kernel [27] provides a platform and a translator. The translator reads GDMO templates and generates program codes running on its platform. Another problem with rich functions is running performance. The rich function requires high performance hardware and, as a result, hardware cost becomes expensive.

The Q3 interface including CMIP was initially assumed established on the OSI protocol stack. The OSI protocol stack does not become popular because of its complex structure, but CMIP over TCP called CMOT [28] is standardized. Some of the concepts of Q3 interface still remain.

The operation system becomes complex for implementing rich functions. It is required to reduce development cost of an operation system. One of the solutions is a component based operation system. CORBA or Web Service is an example of component-based architectures. Service-oriented Architecture (SOA) becomes a notable concept. An operation system developed by Commercial-Off-The-Shelf (COTS), in which an operation system is built by composing commercial products, was proposed [29]. TMF proposes New Generation Operation Support System (NGOSS [30]), which uses an Enterprise Service Bus (ESB) for connecting components.

2-4 Automation of management

Another key for reducing management cost is the reduction of running cost. Saving

administrator efforts is a key because personnel cost occupies large part of running cost. Policy-based management [31][32][33][34][35][36][37][38][39][40][41][42][42][43][44][45] is a trial for saving administrator efforts. A policy is a direction of management without detailed management operations. For example, we have to all INT SERV routers in the path of traffic. It takes a lot of costs to make configurations of them. In the policy management framework [46], a Policy Decision Point (PDP) and Policy Enforcement Points (PEPs) are provided. An administrator gives the PDP some QoS policies. For example, he gives VoIP traffic a privilege. The PDP distributes or replies, in response to the PEPs request, the policies to the routers. The protocol for the distribution is Common Open Policy Service (COPS) protocol. The PEPs interpret the policies and each of them configures itself. It can reduce management efforts.

In 2000s, concept of autonomic computing was proposed. Management operation consists of five functions: Fault management, Configuration Management, Account Management, Performance Management, and Security Management (FCAPS). Autonomic computing proposes four functions except account management: Self-healing, Self-Configuration, Self-Optimization, and Self-Protection [47]. Autonomic Computing consists of three processes:

- A monitoring process in which an operation system watches the situation of managed systems,
- An analysis process in which the operation system checks whether a problem appears in the managed system and suggests what the problem occurs and why, and
- An action process in which the operation system fixes the problem.

In [45], an administrator gives a policy telling a service level. In concrete, it gives target of a response time of a web application. The web server, which is a PEP, tries to keep a service level, by rejecting clients with response of “Server Busy (504)” when it is about to break the service level. Another example of autonomic computing is Self-Organizing Network (SON) [48] discussed in NGNM alliance.

One key of the autonomic computing is the analysis process because correct analysis leads to the correct action by automation. For example, in fault management, fault analysis techniques [14][15][49][50][51][52][53][54][55][56][57] may be one step to fault management automation. It may greatly reduce the administrators’ efforts, and it may be a help for correct automation of trouble shooting in the future autonomic computing era.

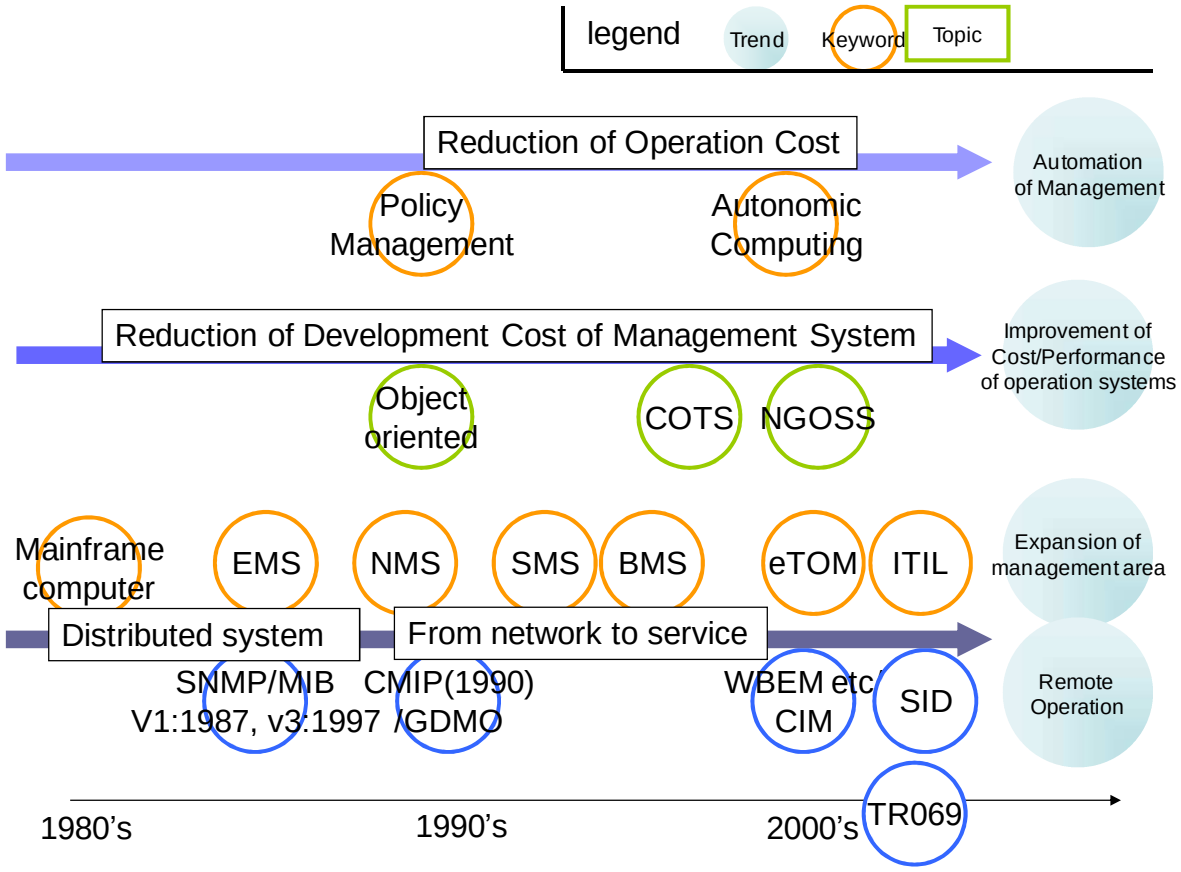

Figure 5: History of Management Techniques of ICT Systems

Chapter 3
High-speed and high functional operation system for OSI Network
– Fast Q3 agent with compact memory MIB –

Chapter 3. High-speed and high functional operation system for OSI Network -- Fast Q3 agent with compact memory MIB –

3-1 Introduction: low cost Q3 agent for telecom access network.

Access network to home has been enlarging in these several years. In the telecom management, the OSI management framework is one of the dominant frameworks. The Q3 interface uses Common Management Information Protocol (CMIP) in the OSI management framework. We have developed a Q3 agent platform embedded in a network element placed in access network. A Q3 agent application program implemented on the platform should running on a single-board CPU board which is compounded in the same rack of a network element. More network elements in the access network run than network elements in the core network. Therefore, the cost of network elements including Q3 agents in the access network is dominant in the whole network system. Using an expensive and high performance CPU board for a Q3 agent have a bad impact on the cost of the whole network systems.

In the telecom network, robustness of Q3 agents is as important as the cost. We adopt, therefore, a flash memory, which have no mechanical structure, instead of hard disk drive. Flash memory is several dozen as expensive as hard disk drive, for the same capacity. It is, therefore, difficult for a Q3 agent to use a huge amount of flash memory.

In addition, CMIP which a Q3 agent uses has rich functions. It requires more complex processing than SNMP does. It becomes overhead of performance. We, therefore, aim at a fast Q3 agent platform with small memory consumption.

3-2 Background: TMN and Q3 agents

As telecommunication services have getting richer functions, network elements and their management functions have getting more complex. International Telecommunication Union – Telecommunication Standardization Section (ITU-T) standardizes the management framework called Telecommunication Management Network (TMN). TMN defines several interfaces between network elements in order to improve the interconnection between elements developed by different vendors. One of the interfaces is the Q3 interface between a manager system and a management agent. Our target is the management agent with Q3 interface.

The management information model of managed network element consists of managed objects (MOs). MO instances (MOIs) are placed in the tree structure called a containment tree. MOIs are stored in Managed Information Base (MIB). An MOI is named after its

position in the containment tree: Its name is a path in the containment tree. It is called a Distinguished Name (DN). Each node in a DN is a pair of Relative Distinguished Names (RDNs). An RDN is a pair of a name attribute and its value.

Seven services are provided for a manager system to access information in a Q3 agent. These services are called Common Management Information Services (CMIS) [58]. For example, the manager system gets the attributes in the management agent with M-GET services. A management agent notifies the manager system of an event asynchronously with M-EVENTREPORT service. Scope operations and filter operation can be used in some of the CMIS services in order to narrowing the target MOIs. ASN.1 [59] is used as data structure of CMIP. Another characteristic of TMN is a Guideline for Definition of Managed Object (GDMO) template, in which the developers of operation systems define managed objects. The developers can define the structure of a managed object with a MANAGED OBJECT CLASS template and the relation between MOIs in the containment tree with NAME BINDING templates. There are totally nine kinds of templates including those two.

3-3 Bottleneck in Q3 agent

The following three are bottlenecks, which prevent from efficient execution of a Q3 agent. We propose architecture of a Q3 agent platform to overcome the bottlenecks.

(1) A Q3 agent is difficult to achieve an efficient retrieval of information of MOIs because the structure of the containment tree is complex. The containment tree admits many types of name attributes. There co-exist many types of attribute values in the containment tree. It prevents from efficient retrieval.

The Q3 agent have to, firstly, find a subnode whose name attribute and its value are equal to RDN. It requires two comparisons: one comparison of name attributes and the other comparison of values of the found name attribute. These two comparisons make retrieve operation inefficient.

(2) The behaviors of MOIs are different depending on its Managed Object Classes (MOCs). Managed Object “log” stores events as MO “logRecord”s in its subordinate nodes. MO log removes a “logRecord” when logRecords are generated more than a given threshold. It is called a “wrap” operation. It is an example that the behavior of managed object under log is different from other managed objects.

(3) The state of network element and the content of MIB have a large gap because

management information in Q3 is abstract. The transformation between the state of network element and the contents of MIB needs heavy load. We proposed, in [60], a bitmap approach in which the state of network element is directly transferred to the MIB without transformation. It is an efficient approach because of no transformation in the transferring from NE and MIB. It takes time when managed system gets an attribute of a managed object from MIB because the transformation occurs in the bitmap approach.

3-4 Agent architecture

Figure 6 shows the architecture of a Q3 agent including the agent platform we propose.

- **Agent Platform (Agent PF)**: It is a software infrastructure of the agent. It provides the following functions which are the basis of Q3 agents.
  - ✧ Association management function, which responds to association requests and termination requests from a manager system.
  - ✧ CMIS processing function, which accepts CMIS requests and provides the requested services.
  - ✧ Event reporting function, which reports, to the manager system, events detected by NEAEs or NE.
  - ✧ Event-driven function, which is invoked when an event occurs. We implement MO “log”, which accumulates MO “logRecord”s, using this function.
  - ✧ Periodical invoking function, which is invoked at regular intervals. We implement MO “historyData”, which is a log of MO “currentData”, using this function.
- **Agent Application (Agent AP)**: Agent developers implement NE-specific functionality as agent applications.
- **MIB Platform (MIB PF)**: The agent platform uses main memory MIB instead of a secondary storage device such as a hard-disk.
- **ISODE** [61]: Libraries that implement communication protocols from RFC 1006 [62] up to CMIP.
- **NEAE 1 and NEAE 2** [60]: NEAE stands for “NE access engine”. NEAE 1 exists in the agent, and NEAE 2 is in the NE. These two communicate with each other using the UDP/IP to accomplish the following two tasks.
  - ✧ They update the contents of MIB, referring to the state of NE.

✧ They detect events in NE and report the events to the agent.

We often regard these two modules as one module and call it NEAE.

- **Lynx OS**: It is a real time operating system based on POSIX.
- **NE**: ATM OC3 package is an example of NE.
- **Manager System**: It is used by administrator in order to watch the state of NEs and the events issued. It has the Q3 interface for communication with a Q3 agent.

Agent PF, Agent AP and NEAE are written in C++ programming language.

A rectangle with dashed lines in Figure 6 is a Q3 agent. A rectangle with chain lines is an agent platform. We developed the MIB PF, NEAE1 and NEAE2.

We describe the MIB PF in this section because the data structure in MIB is a key to a memory-saving and a fast Q3 agent. We propose three issues in the MIB PF architecture.

(1) We propose compact data structure of the containment tree. The data structure enables a fast access to MOI.

(2) The containment tree architecture accommodates different types of managed objects. We can choose suitable data structures of managed objects in the containment tree.

(3) We provide several stored architectures, which co-exist in MIB, for several kinds of attributes. The developers can choose a stored architecture to each attribute, considering the frequency of update of attributes.

A developer can choose an architecture component suitable to each managed object or each type of attributes of managed objects and, as shown in (2) and (3) GDMO templates are good hints for the developer to choose a suitable one.

Our agent platform is designed with object-oriented design pattern framework [63] in order that several architecture components coexist in a Q3 agent. An agent application can be implemented on the agent platform regardless differences of these architecture components.

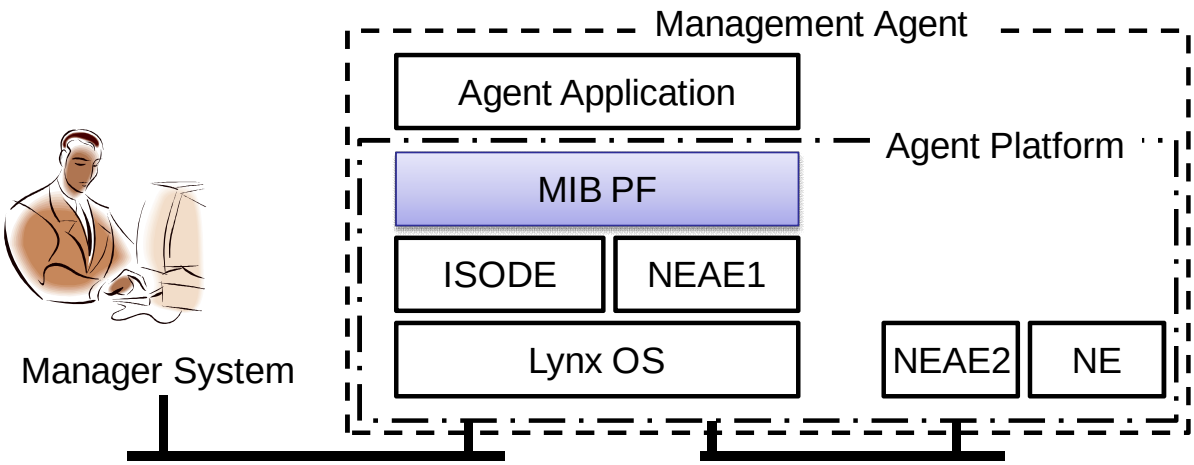

Figure 6: Modules in the architecture

3-4-1 Directory structures in containment tree

A node in the containment tree in our Q3 agent platform consists of two types of data structures: A directory structure is an array of name attributes, and directory substructures are arrays of values of the name attribute. A node of the containment tree is shown in Figure 7. A node corresponds to an RDN of the DN indicating an MOI. It consists of a directory structure and some directory substructures.

A directory structure is a table of pairs of a key and a pointer. The key is a name attribute and the pointer points to directory substructure. The developer can define, in the design phase, the name attributes of a directory structure, referring to the NAME BINDING template of the managed object class store in the node.

A directory substructure is a table of tuples of a key and two values: The key is a name attribute value in the directory structure, and the values are a pointer to an MOI and a pointer to a subordinate subtree if necessary.

Steps to find an MOI of the given RDN are the followings:

(1) The platform finds, in a directory structure, an entry whose key is equal to the type of the name attribute of RDN.

(2) The platform accesses the directory substructure to which the pointer in the found entry points. The platform, then, finds the entry in the substructure whose value is equals to the value of the RDN.

The separation of a directory structure and its directory substructures has three advantages.

Firstly, this contributes a first retrieval of the data. In Step (1) the platform finds the directory substructure from the entries of the directory structure whose name attribute is the same with that given in the RDN. This step narrows the scope of searching, and, in Step (2) the platform only compares values in the directory substructures with the value in the given RDN without considering the difference of the types of the name attributes.

Secondly, this also contributes to reduce the memory size of the containment tree. This is because a node need not store the all the attribute types. If a node was stored in a pair of the type of a name attribute and the value of the name attribute, the platform would have to store the types of name attributes redundantly. Consider a node stores *m* MOIs with the *n* types of the name attributes. Our platform stores totally *n* entries in a directory structure and *m* entries of each directory substructures. In short, there are *n* entries for name attributes, while (*n m*) entries exist in the node.

Thirdly, three different types of MOs can coexist in the same node. Three types of directory substructures under the same super class are provided in the platform. The directory structure can store these three types without distinction. This is mentioned in the next subsection.

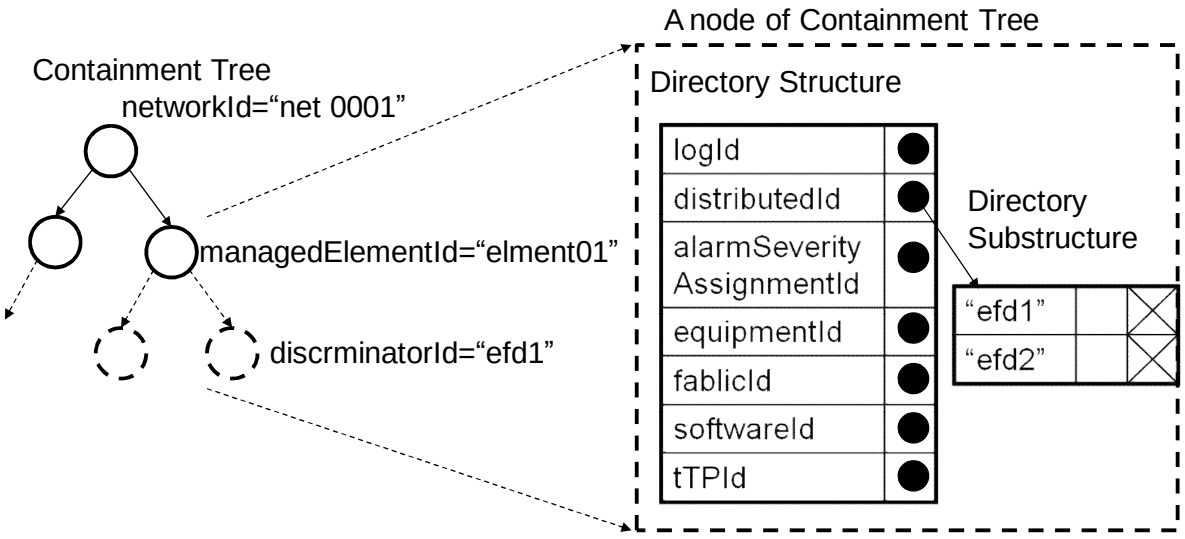

Figure 7: Directory structure and directory substructures

3-4-2 Three types of directory substructures

Each node in the containment tree behaves differently from one another, depending on the types of MO stored in the node. Wrap operation mentioned in Section 3-2 is an example. We provide, in the platform, three types of directory substructures.

- **Static directory substructure**: Some MOs cannot be removed or generated in runtime. They are provided a priori, and the number of them is fixed. We use a static directory substructure for this kind of MOs. We implement it in a fixed-size array. The platform can firstly find an MOI with a hash function. The developer can identify this type of MO in the design phase, referring to CREATE/DELETE keyword in the NAME BINDING template. If no CREATE or DELETE keyword is not defined for the MO, the developer can decide to use the static directory substructure.
- **Dynamic directory substructure:** We use a dynamic directory substructure for MOs which can be created or deleted in runtime. A dynamic directory substructure is implemented in a linear list. An MOI can be inserted or deleted in the linear list when it is created or deleted. We use a linear search in this directory substructure, and it is slower than the hash-function search in the static directory substructure.
- **Ring directory substructure:** It is used for the wrap operation. It is implemented in a fixed-size ring buffer. The wrap operation is effective in this structure. For example, consider that new “eventRecord” is generated in a ring buffer with m entries. The buffer is assumed to be already full. The oldest MOI stored in entry is removed, and the generated “eventRecord” is inserted instead.

3-4-3 Three attribute representations

Because attributes of MOs have different characteristics, storing the attributes in adequate data representation in MIB leads to fast access to data and small memory consumption [69]. We have classified the attributes into the following three categories.

- **Normal attributes**: logical data. *Logical* means that it does not directly reflect the state of the NE. For example, an RDN attribute is a normal attribute.
- **Shared attributes**: Attributes which have the same value among multiple MO instances. The platform provides a facility that the MO instances can share such attributes. Therefore, we can save memory space [64]. The example is the “objectClass” attribute defined in MO “top”. All classes inherit the attribute. If MO instances are generated

from the same MO class, “objectClass”-s have the same values. The MO instances can share the same attribute.

- **Resource attributes**: Resource attributes directly reflect the state of the NE. NEAE transmits the state of NE into resource attributes periodically (i.e. polling).

  In the platform, the format of a resource attribute is just a bit-pattern sequence which is a copy of the values in the memory or the register of the NE. We call it a “native NE format." The native NE format is difficult for the agent platform to access and interpret because it is not in a C++ data format. When the platform accesses a resource attribute, format conversion from the native NE format to the C++ data format is necessary.

  This architecture, however, has two advantages.

  (1) NEAE can transmit the state of NE fast because it need not convert the data format. NEAE simply copies the bit sequence of the NE memory or registers into main-memory MIB. Conversion only occurs when the platform accesses the attributes. It is effective when the accesses are few.

  (2) The area for storing resource attributes is usually smaller than in the case where resource attributes are stored in the C++ data format. For example, an integer value less than 255 may be stored in 1 byte in the NE native format, whereas, 4 bytes are necessary in the C++ data format.

3-4-4 Class information table

We focused on accelerating the operation for getting attributes from an MO instance. For example, we have to access attributes of many MOs gotten by a CMIS scope operation in order to test a CMIS filter. Therefore, the efficiency of this operation has an impact on the total performance.

For access to attributes of MOs, we provide a class information table and attribute information tables. The contents of the latter are the structure of an MO class, and the contents of the former are the correspondences of an MO class and an attribute information table. Because we can get the position of an attribute in a MO class quickly using those tables, we can access the attribute in high performance.

Figure 8 shows the class information table and attribute information tables of the agent platform. We access an attribute of an MO instance by the following process.

(1) We get the internal class tag of the MO instance. An internal class tag is assigned to a MO class.
(2) We get an attribute information table to which the tag points.
(3) We get an offset in the entry of the attribute information table, by using an attribute ID.
(4) We can finally get the attribute from the MO using the offset.

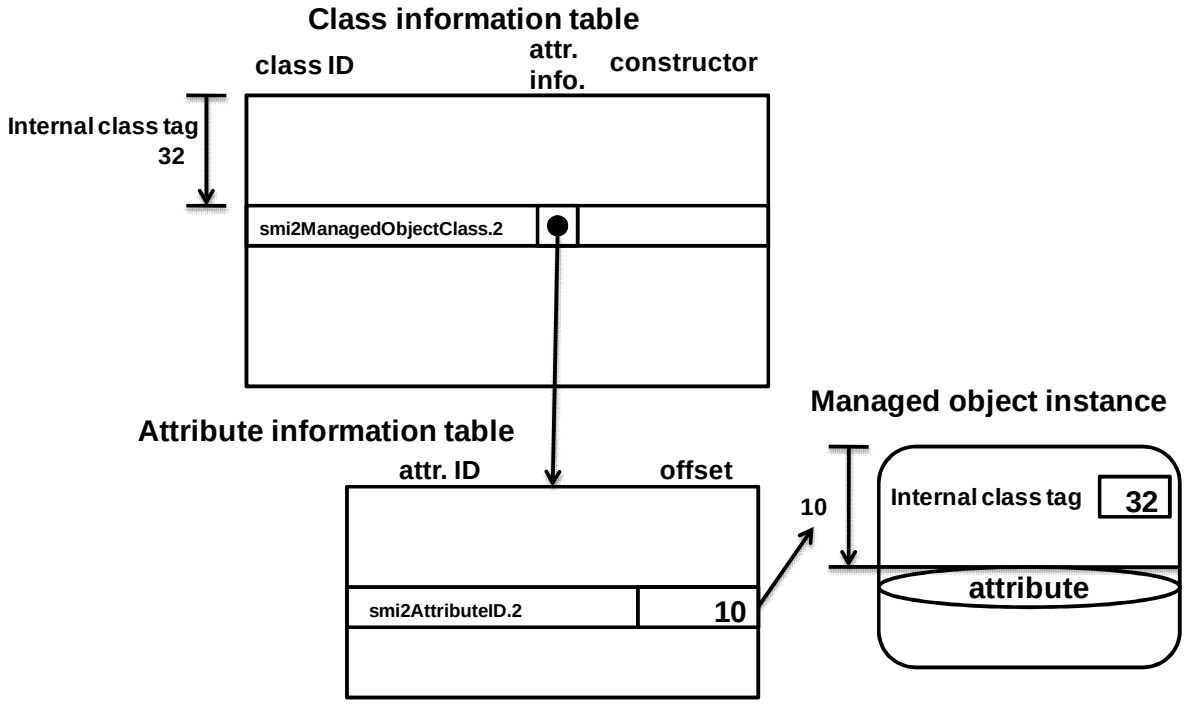

Figure 8: Class information

Let us compare the class information management with that of OSIMIS [65]. OSIMIS has a kind of a cache mechanism, which provides effective attribute encoding: Once an attribute is encoded into an ASN.1 presentation element, the encoded element is cached and, as a result, we can avoid encoding again. Our agent platform uses this effective OSIMIS module.

Figure 9 shows how OSIMIS manages the class information. An attribute information table is a list of fragments of attribute information. Each fragment corresponds to a set of attributes and is additionally defined in a subclass definition, and an attribute information table gathers the fragments corresponds to all attributes of a class.

The advantage of the OSIMIS management is that subclasses can share fragments of

attribute information of their super class. This means memory consumption is small, but, it takes time to get an attribute because we have to navigate through several pointers among the fragments.

In contrast to OSIMIS, our class information is very simple. All attribute information of each MO class is gathered into an attribute information table. Thus, we can use a hash function to retrieve the attribute information quickly.

The memory requirement for attribute information in OSIMIS may be smaller than that of our platform, but we estimated it to account for less than 2.5% of the MIB. Therefore, the reducing the data size of attribute information is not effective. We thought that our data structure of class information should aim at fast retrieval.

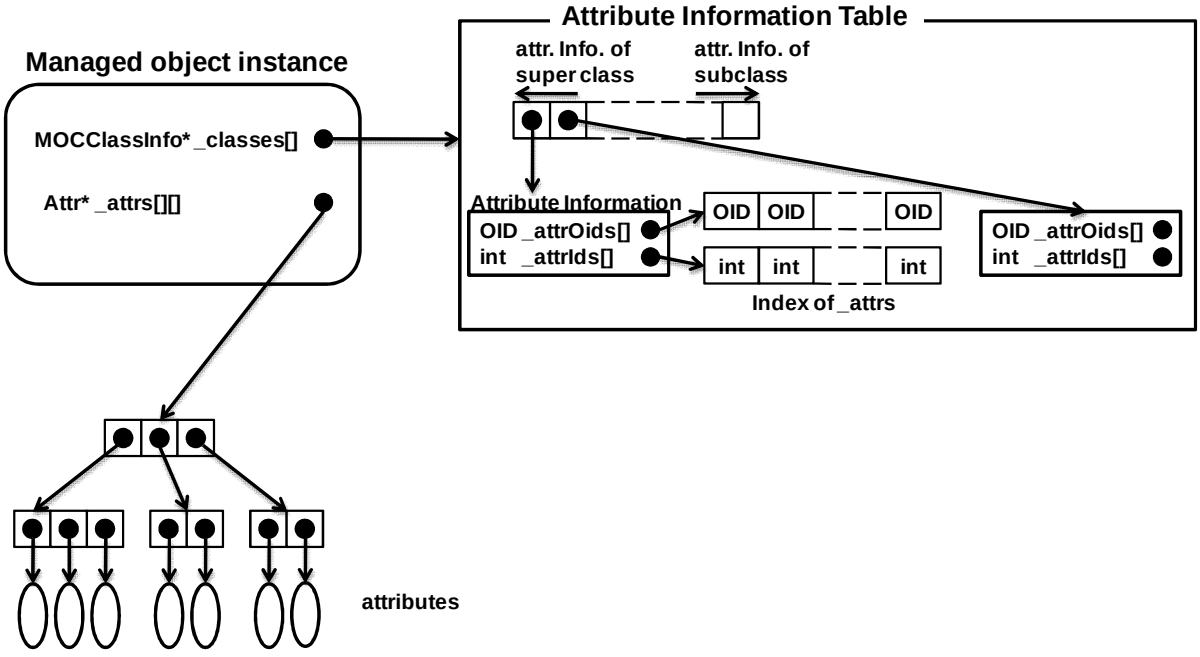

Figure 9: Class information of OSIMIS

3-5 Performance measurement

We evaluate the performance measurement in the following two view points.

(1) We measured the response time of each CMIS service. It is for measuring a total performance of each service.

(2) We measured the process size and the response time of each architecture component.

We can evaluate how effective the selection of architecture components can be.

3-5-1 Measurement method

We evaluated performance in the following environments:

- **System for evaluation**: We installed a Q3 agent implemented on the proposed agent platform running on a work station and a simple manager system on another work station. These two workstations are connected with 10baseT. We used a pseudo software network element. The performance of the workstations are following:
  - ✧ **CPU:** Ultra Sparc 1 (143MHz)
  - ✧ **On-Chip Cache:** 16kB
  - ✧ **Second Cache:** 512kB
  - ✧ **SPECint92:** 215
  - ✧ **Memory:** 96MB
- **Model of NE:** We implemented a simple model based on M3100 [66]. Figure 10 shows the containment tree. Right side in each node shows its name attribute and its values.
- **Response time:** We measured the response time from a CMIS request to a CMIS response with a protocol analyzer.
- **Process size:** We used UNIX *ps* command in order to measure the size of virtual memory. We regard it as a necessary memory size.

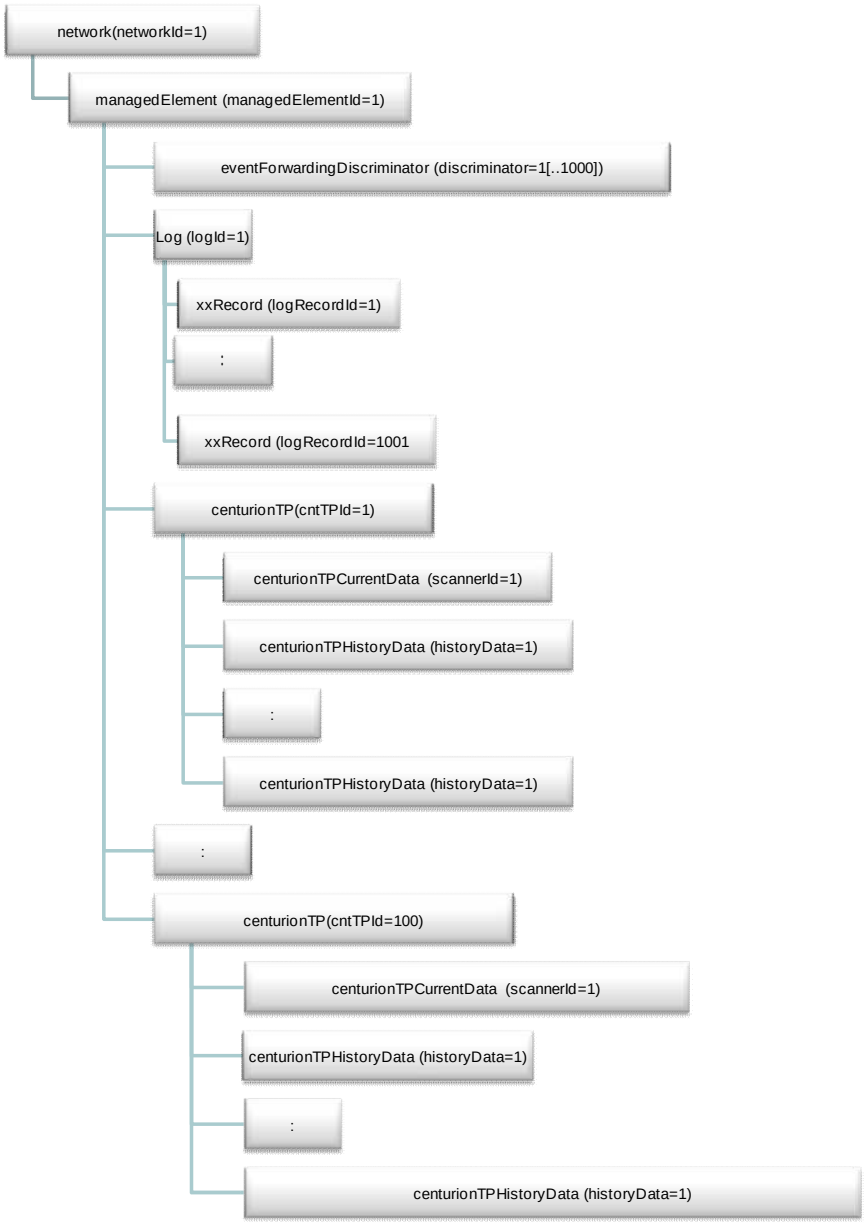

Figure 10: A containment tree for the evaluation

3-5-2 Measurement results

3-5-2-1 Basic performance

Response times of CMIS operations in the proposed PF and OSIMIS [65] are shown in Table 1. The “baseObject” scope without filter is used for this evaluation. The operations but

M-ACTION and M-EVENTREPORT are done on MO “eventForwardingDiscriminator” (eFD in the containment tree shown in Figure 10. The M-ACTION is done on MO “managedElement”, and it records a log into a file. The M-ACTION is without ACTION-INFO and ACTION-REPLY.

The CMIS operations are done on the third level from the root of the containment tree in the evaluation of the proposed platform, but they are done on the second level in the OSIMIS. It is because the containment trees for evaluations are different in the proposed platform and OSIMIS.

Table 1 shows that the proposed platform is faster than OSIMIS in all CMIS services. Performance of M-EVENTREPORT is shown in Table 2. We evaluated the case when “logRecord” is recorded under MO “log” and when “logRecord” is not recorded. We measured the elapsed time for hundred events which the pseudo NE issued and calculated the average time for an event. The M-EVENTREPORT is issued from MO “managedElement” and is forwarded through an “eFD” without filter to a manager.
We evaluate the memory size for an MOI. We measured process sizes, changing the number of MO “eFD”. The results are shown in Table 3. MO “eFD” is implemented in 1.81kB in the proposed platform while 1.14kB in OSIMIS. As described in Section 3-4-4, MOIs of the same class shares some attributes in their super class. It may cause an efficient memory usage.

Response times of M-GET with scopes are shown in Table 4. Column “#MOI” is the number of MOIs found in the M-GET. Column “First Reply” is a response time by a first linked reply. Column “Avg. of next reply” is an average reply time of the rest linked replies. Column “Average” is an average response time of the whole replies to an M-GET request.
Reply times of the “Avg. of next reply” vary widely because of the flow control TCP. The traffic congestion was caused by many packets of the linked replies, and it caused TCP flow control packets.

The response times of M-GET with filters are shown in Table 5. The base object (i.e. the root object of target subtree) of the M-GET is MO “network” with Scope “wholeSubtree”. Column “#MOI under baseObj” is the number of MOI under the MO “network” in the containment tree. Column “#MOI” is the number of MOIs found in the M-GET. Column “Avg. response time” is an average response times of whole replies.

Table 1: Response times (msec) of CMIS operations

	M-GET	M-SET	M-ACTION	M-CREATE	M-DELETE
PF	1.69	1.65	1.70	1.83	1.93
OSIMIS	1.85	1.81	N/A	1.99	2.04

Table 2: performance of M-EVENTREPORT

	Average elapsed time (msec.)	Performance (events/sec.)
With logRecord logging	6.3	158.5
Without logRecord logging	5.6	177.4

Table 3: Process Sizes (kB)

#eFD	0	100	500	Average
PF	3,648	3,816	4,552	1.81
OSIMIS	3,112	3,272	3,668	1.14

Table 4: Response times (msec.) of M-GET with scopes

Scope	#MOI	First Reply (msec.)	Avg. of Next Reply (msec.)	Average (msec.)
baseOnly	1	--	--	2.4
firstLevel	1	9.0	39.4	46.4
indivisualLevel	12	7.5	5.4	6.0
baseToNth	14	5.0	2.5	2.0
wholeSubtree	444	9.0	1.8	5.1

Table 5: Response times (msec.) of M-GET with filters

Length of filter	#MOI under baseObj	#MOI	Avg. response time (msec.)
0	124	124	4.6
1	124	100	5.9
2	124	100	6.3
3	124	100	6.4

3-5-2-2 Performance and memory usage in the cases of attribute types

The response times to different types of attributes are shown in Table 6. These response times are almost the same. There are no explicit difference between the response time of a normal attribute and a resource attribute. It means that we can successfully reduce the overhead of transformation from the state of NE to an attribute which CMIP can handle.

We compare memory usages between the case when Attribute "objectClass"s of MO "eFD" is implemented in shared attributes and the case there they are implemented normal attributes. We measured process sizes, changing the number of MOI "eFD"s. Column "#MOI" is the total number of MOI in the containment tree. Column "Average" is an average memory usage for an MOI. Row "Reduction (%)" is the ratio of saved memory usage of shared attribute over that of normal attribute. The result is shown in Table 7. It shows that memory usage gets efficient when the "#MOI" increase. This is because the shared attributes increase when the "#MOI" increases.

Table 6: Response times (msec.) for attribute implementations

	Normal Attribute	Resource Attribute	Shared Attribute
M-GET	2.4	2.6	2.1

Table 7: Process sizes (kB) with/without shared attributes

#MOI	24	5,024	10,024	30,024	Average
Normal Attr.	3,648	13,272	22,896	61,296	1.92
Shared Attr.	3,648	12,600	21,536	57,296	1.79
Reduction (%)	0	5.06	5,94	6.53	

3-5-2-3 Performances in types of directory substructures

We measured the performance improvement by the three types of directory substructures. We use M-CREATE, M-GET, and M-DELETE for the performance evaluation.

We measured an average response time in the case when M-CREATEs were issued 100 times and 100 MOIs were created. The 100 MOI spaces for the created MOI were reserved in the static directory substructure and in the ring directory substructure.

We measured an average response time of M-GET with Scope "baseObject" and without filter. The response time in the dynamic directory substructure varies depending on the

position in the linear list. We, therefore, evaluate the response time of MOI in the middle of the linear list.

We measured the response time of M-DELETE by deleting the 100 MOI created by the M-CREATE. The platform moves MOIs in the ring buffer subdirectory in order to place MOI in the continuous area in the ring buffer when an MOI is deleted (compaction). The compaction impacts on the performance. We, therefore, randomly choose an MOI to be deleted.

The evaluation results are shown in Table 8. M-CREATE in ring buffer subdirectory is faster than other two. This is great improvement of performance because creations of MOI "logRecord" occupy total MOI creations in Q3 agent. The performance of ring-buffer substructure in M-DELETE is not largely different from other two. The overhead of the compaction is not so large.

Table 8: Response times (msec) for directory implementations

Directory substructure	M-CREATE	M-GET	M-DELETE
Static	3.3	2.7	2.4
Dynamic	3.5	2.8	2.4
Ring Buffer	2.3	2.7	2.7

3-5-2-4 Performance evaluation by using ATM OC-3

We also evaluated the platform connected to an actual NE, while pseudo NE is used in the previous evaluations. We used ATM (OC-3) and implemented a Q3 agent on the single-board computer with PowerPC. Figure 11 shows the evaluation environment.

The evaluation results are shown in Figure 11. Row "Sparc" is an evaluation result under the environment mentioned in the previous sections. The elapsed time of M-EVENTREPORT in the proposed PF on Power PC is three times slower than that in Sparc. It is because OC-3 is a bottleneck. It cannot send events to the Q3 agent faster. In other word, the Q3 agent has enough performance for this OC-3. We also used a protocol analyzer instead of the OC-3 in order that NE sends events faster. In that case, the Q3 agent can send M-EVENTREPORTs more than 150 events/sec.

The result shows that the response time of M-SET is slow. In this evaluation, the

attribute for the M-SET is placed on OC-3 instead on MIB, and the platform directory updates the attribute. We call this kind of attribute an on-demand attribute. The agent requests an NEAE to update an on-demand attribute. This takes time of M-SET on an on-demand attributes.

Table 9: Performance (msec.) on Power PC board

	M-GET	M-SET	M-EVENTREPORT
Sparc	4.9170	--	6.309
Power PC	6.165	162.8	21.82

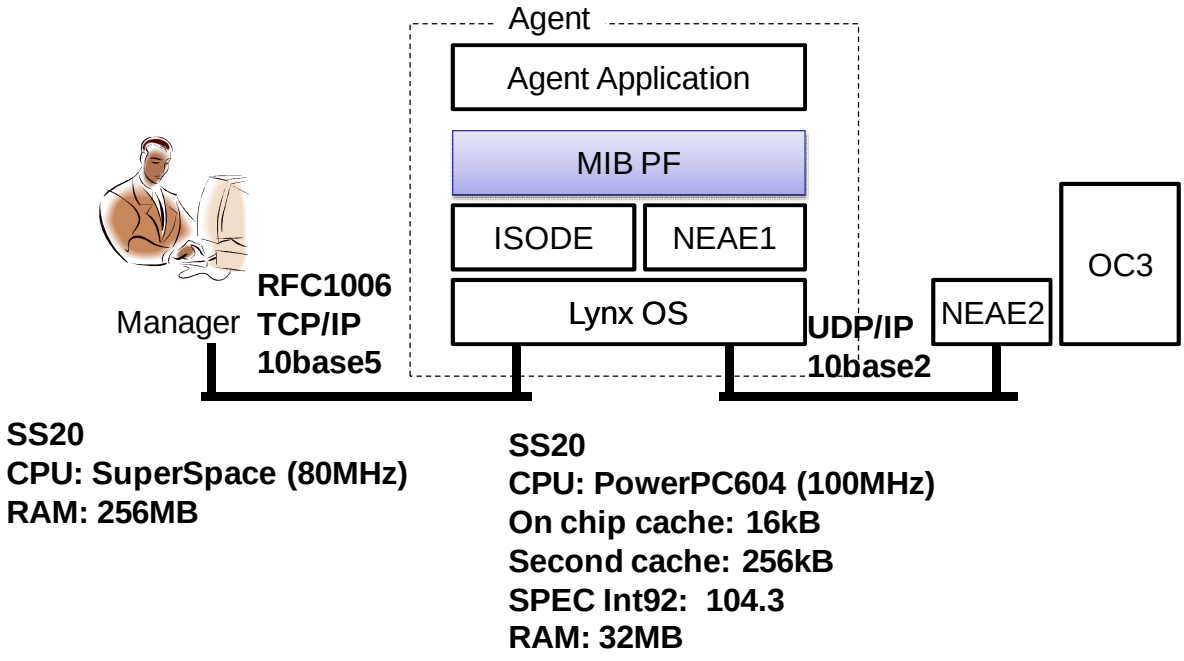

Figure 11: Agent on PowerPC connected with OC-3

3-5-3 Discussion

There are many products for TMN management platforms [27][65][67][68][69][70][71]. The OSIMIS is open source software of TMN agent platform. We discuss the comparison between the containment tree in the OSIMIS and the proposed platform.

The containment tree in the OSIMIS is shown in Figure 11. A node consists of a linear

list (i.e. _peer) which connects MOIs in the same level of the containment tree, a pointer to its superior node (i.e. _superior), and a pointer to its subordinate node (i.e. _subordiate). To find a subordinate node, the OSIMIS has to navigate a linear list in _peer. Assume that there are k types of name attributes and each type of name attribute has n MOIs. OSIMIS navigates a _peer linear list (kn / 2) times. Our platform navigates links to a subordinate node 2 times in the static directory substructure. It navigates links (1+n / 2) times. The OSIMIS takes time more than the proposed platform.

A linear hash is used in [70]. A linear hash can dynamically extend and reduce a size of hash table, and MOI can, therefore, be generated and be deleted in the linear hash table. A linear hash is flexible in its size, but its size could be only 2^n, where n is integer. It is inefficient in memory usage. Tree data structure as well as a linear hash is also used in [70], and it is inefficient in memory usage, too.

We discuss the advantages of three types of directory substructures for types of MOs. The ring buffer substructure provides a first retrieval as shown in Table 8. The ring buffer substructure is used for storing MO “logRecord”-s. For example, when a fault in NE occurs, a lot of events occur at one time. A lot of MO “logRecord”-s are, therefore, generated. It is required for a Q3 agent platform to create MO “logRecord”-s and to store them efficiently. The ring buffer substructure is suitable for “logRecord”-s because of the first creation shown in Table 8.

We discuss advantages of implementation methods based on types of attributes. We measured the performance shown in Table 7. Shared attributes contribute to memory saving. We will use Attribute “packages” and Attribute “nameBinding” as shared attributes as well as we will use Attribute “objectClass” as shared attributes.

There are several works where several architecture components are provided for several types of attributes, as shown in this thesis. [71] classifies four types of attributes: a static attribute, which changes its value, a pseudo attribute and dynamic attribute, which are not updated by network manager but which may changes, and GET-SET attribute, which can be updated with M-SET. The system in [71] manages a locking mechanism considering the types of attributes. It improves the performance with parallel processing.

Finally, we discuss the data translation between NE and MIB. Performance of resource attributes but on-demand attributes are almost the same with that of normal attributes as shown in Table 7. A first transfer of resource attributes is expected because no

transformation is necessary as mentioned in Section 3-4-3.

An agent application program can be implemented independent from the differences of on-demand attributes and resource attributes. The NEAE translation function provides the same operations to the application program. It accesses an on-demand attribute on NE through the NEAE, while it access a resource attribute in the bitmap of MIB.

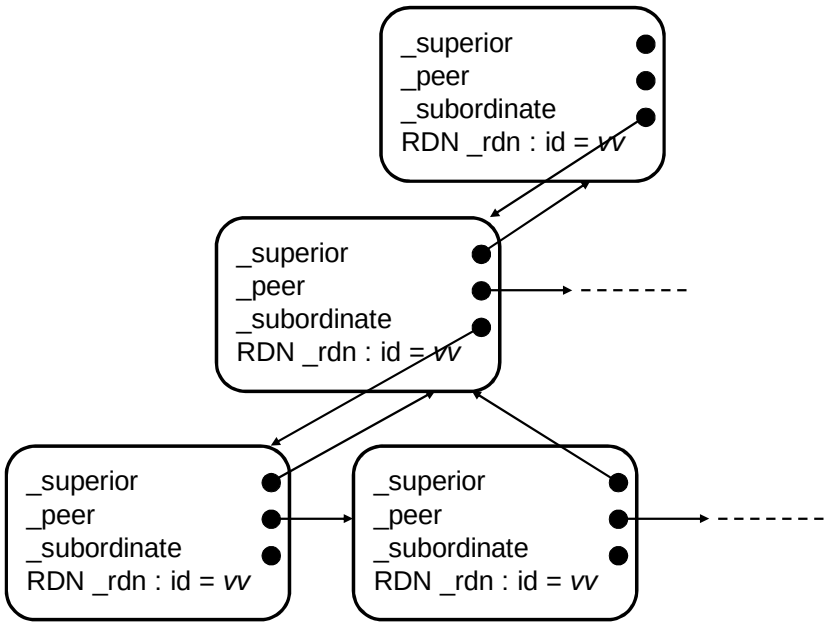

Figure 12: Containment tree data structure of OSIMIS

3-6 Conclusion

We proposed a Q3 agent platform embedded in NE. The platform provides several architecture components which is chosen based on the types of MO and the types of attributes. It overcame the bottlenecks of the Q3 agent and it achieved first operations and small memory usage. The developer of Q3 agent applications can choose suitable architecture components, considering the information model written in GDMO templates.

As shown in Table 7, this platform requires 22MB for 10,000 MOIs. The Q3 agent on platform storing 30,000 MOIs requires 60MB. It cat not be installed on the Power PC board with 32MB memory mentioned in Section 3-5-2-4. OSIMIS is also difficult to be installed in the Power PC board. The attributes encoded in ASN.1 occupies a large part of the MIB. ASN.1 compiler Pepsy [65] is used both in the platform and in OSIMIS. Almost the same

attribute sizes occupies in both platforms. We have to implement an ASN.1 compiler which can generate memory-saving attributes.

The memory cost becomes cheaper now, but the memory cost is still dominant in CPU board. The management information has been getting large because management functions are getting rich. The memory reduction effort is, therefore, still necessary now.

Chapter 4
Reducing development costs and maintenance costs for operation system
– DSL approach for reducing affected maintenance cost –

Chapter 4. Reducing development costs and maintenance costs for operation system – DSL approach for reducing affected maintenance cost --

4-1 Introduction: Reduction of Development Costs and Maintenance cost

We developed the Q3 agent platform mentioned in Chapter 3. It can run on an embedded system with small capacity of memory. Q3 agents running on the platform is embedded in a lot of network elements placed on access networks. The reduction of memory size has a good impact on total costs.

We have to pay attention to another issue for reduction of the capital investment cost of operation systems. It is a development cost of the operation systems. We have to develop a Q3 agent application program on the platform for each network element. In addition, if the network element is versioned up, an operation system must be updated, too.

There are so many approaches for software development [72]. One approach to reduce application development costs is a domain specification language (DSL). A DSL is a kind of programming language tailored to a specific domain. It provides developers with syntax features which are suitable to the specific domain. It helps the developers to easily specify the specifications of applications. A translator of the DSL translates the specifications into program codes. It improves productivity of application programs. It is known that reuse of the specifications written in DSL can improve the reusability more than reuse of programming codes [73].

We have to also consider maintenance cost of existing application programs as well as development costs for new application programs. Because network elements and operation systems which manage the network elements will be used in long time, the maintenance of them is important. In the maintenance, understanding the requirements and the structure of the application programs takes more costs than testing and repairing the application programs [74]. In the DSL approach, the developer can more easily understand the requirements and the structure of application program by reading the specifications written in a DSL than program code written in a programming language. A DSL translator can automatically generates a new program code from maintained specifications written in a DSL

Maintenance is classified into three as the following [75]:

- **Maintenance for repairing** for fixing defects,
- **Maintenance for adaptation** for adapting a program into new running environment, and
- **Maintenance for perfectness** for extension of functions and satisfaction for changes of requirements.

Chapter 4. Reducing development costs and maintenance costs for operation system – DSL approach for reducing affected maintenance cost --

We add another class of maintenance in the DSL development approach called **Maintenance for affected modules** or **Affected Maintenance**. We have to maintain a DSL translator in accordance with updates of applications. For example, consider that an application generated with a DSL translator is planned to be imported to another operating system. We have to maintain a DSL translator in order to generate a program code for the new operating system. Another example of the maintenance for affected modules is the case that a new function is extended in the platform where the application runs. We have to add a new syntax feature into the DSL in order to use the new function, and we have to also extend the DSL translator so that it can accept the new syntax feature and it can generate program code for the function.

Maintenance of DSLs, including maintenance for affected modules, is required as well as the maintenance of application program. It results in reduction of total development and maintenance costs. Development environments and methodologies for developing a DSL and its translator are required. We propose, in this chapter, a DSL toolkit, which can improve the productivity and the maintainability of DSL translators.

4-2 Related work

A DSL and its translator improve the productivity of software. For example, a "Form-based Service" system can be easily developed in Mawl [76]. A form-based service system processes with response to users' inputs. The typical example of form-based service systems is a web application. Mawl also improves the maintainability of developed form-based sevice systems. Mawl provides a type check mechanism to the inputs from users. It improves maintainability of systems because the type check mechanism excludes the risk of writing logically incorrect programming codes.

Generally, a DSL and its translator can improve the maintainability of software, but maintainability strikingly decreases when DSL itself should be maintained [77]. Therefore, a DSL toolkit is important for developers to improve the productivity and maintainability of DSL [78]. It can reduce the cost of the maintenance of affected module.

ASF+DSL [77] consists of a context-free grammar matching (CFGM) system and a Term Rewriting System (TRS). It can easily develop a DSL translator: CFGM parses the DSL and TRS rewrites the specification in DSL into program codes. The developers of a DSL can formally specify the translator with the algebraic specification. However, TRS systems are

difficult to be debugged. For example, in usual programming, the developer runs a procedure in program codes as a test. It is useful for localizing a problem in the program codes. However, in TRS system, activating some of rules does not run correctly.

Draco [79] supports reuse of specifications written in a DSL, design, and programming codes, and it improves the productivity and the maintainability of software. Draco defines mapping between DSLs in different abstraction levels. It translates DSL, through these mappings, into program codes step-by-step. Draco manages the mapping among several DSLs, but it does not provide a mechanism for a new DSL. It, therefore, does not support maintenance for affected module, such as the change of syntax of a DSL [80].

Stage [78] provides a template which defines generated program codes. The template consists of dynamic parts and consistent parts. Stage directly generates a program codes as the consistent parts. The dynamic parts include processing codes accessing a concrete syntax tree expressing input of the DSL. However, for the simplicity, Stage does not have a mechanism to generate a recursive programming codes. For example, mathematical formula written in DSL has inherently recursive structures. Generated codes should include recursive structure in some domains.

KHEPERA [81] translates an abstract syntax tree into another abstract syntax tree several times. It finally prints out the final abstract syntax tree into text form. Developers of a DSL can write the translation rule in a language which KHEPERA provides. KEPERA also provides a debug function. It shows the relation between input specifications and generated codes. This is useful for DSL users because they can easily trace the input specifications when they want change the generated codes. This is also useful for the DSL developer because they can easily trace the translation process and they may find a problem in mapping rules. However, it is difficult for them to find a bug because of several mapping rules. The developer can understand plural steps of mapping. It is not easy task.

We overlook the previous work, and we summarize requirements of DSL toolkits as follows:

- The developer of a DSL can easily find a place in a DSL translator to be modified for fixing a problem,
- They easily guess the places of generated codes are modified, when they change the DSL translator,
- They can test a part of functions of the DSL translator for localizing a problem in the

DSL, and

- The DSL translator can generate programming codes expressive enough for many domains. For example, it can, of course, generate program codes with recursive structures.

4-3 Rosetta: A DSL toolkit

We developed a DSL toolkit called Rosetta. It generates a DSL translator written in Java. It helps DSL developers with Java knowledge.

4-3-1 Architecture of Rosetta

Figure 13 shows the architecture of Rosetta. A large rectangle including some components is a DSL translator generated with Rosetta. Rosetta provides hatched rectangles in the figure. They are software libraries or off line tools. The DSL developers develop modules expressed as rectangles with underlined text. They also specify specifications expressed as lozenges with underlined text.

A DSL translator developed with Rosetta translates input specifications into a program codes as following.

A) A parser reads specifications written in a DSL, parse it, and generates an abstract syntax tree (AST). The generated AST is stored in an AST database.

B) A semantic checker access the AST in the AST database, and it checks whether the specifications are semantically correct. The semantic checker terminates the translation or continues it with error messages when it finds a semantic error.

C) A code generator accesses the AST in the AST database, and generates program codes, by referring output templates. The output templates are kinds of scripts. They define how the generated codes are.

The usual method for development of a DSL translator is shown in Figure 14. The developers define the syntax of a DSL in BNF. They use a parser generator, which generates a parser program of the DSL. The generated parser program codes are skeleton codes. The developers have to fill codes in action parts in the generated parser program. For example, they fill program codes for creating an abstract syntax tree. The developers also write a program code for the code generator. For example, they use the visitor design pattern [63] for the code generator.

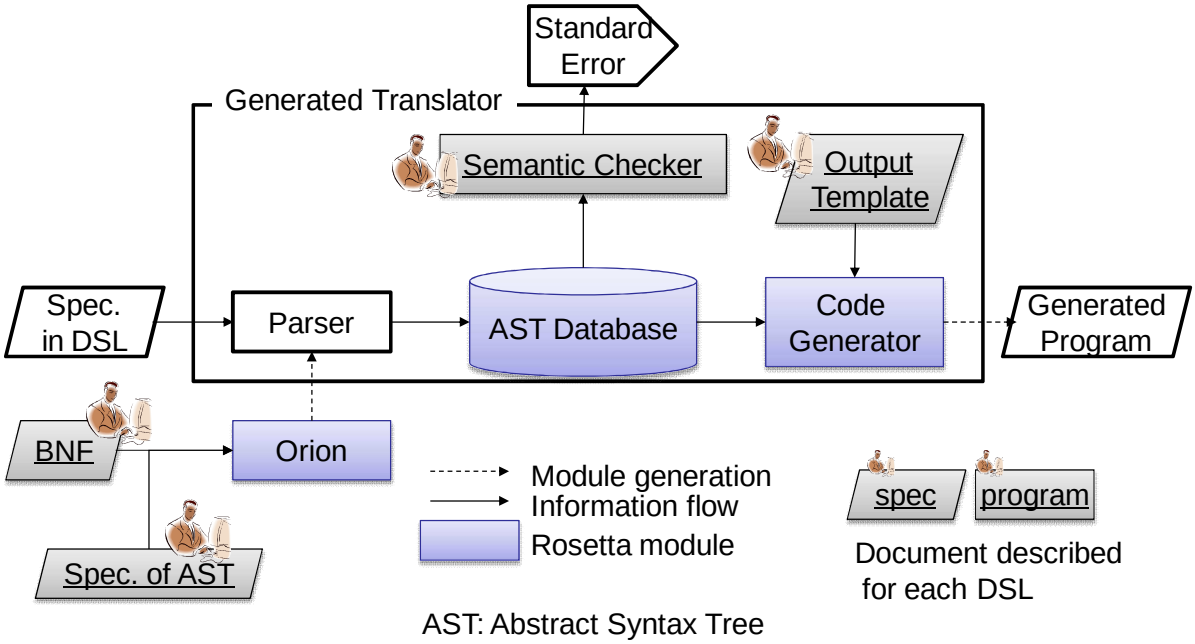

Figure 13: Architecture of Rosetta

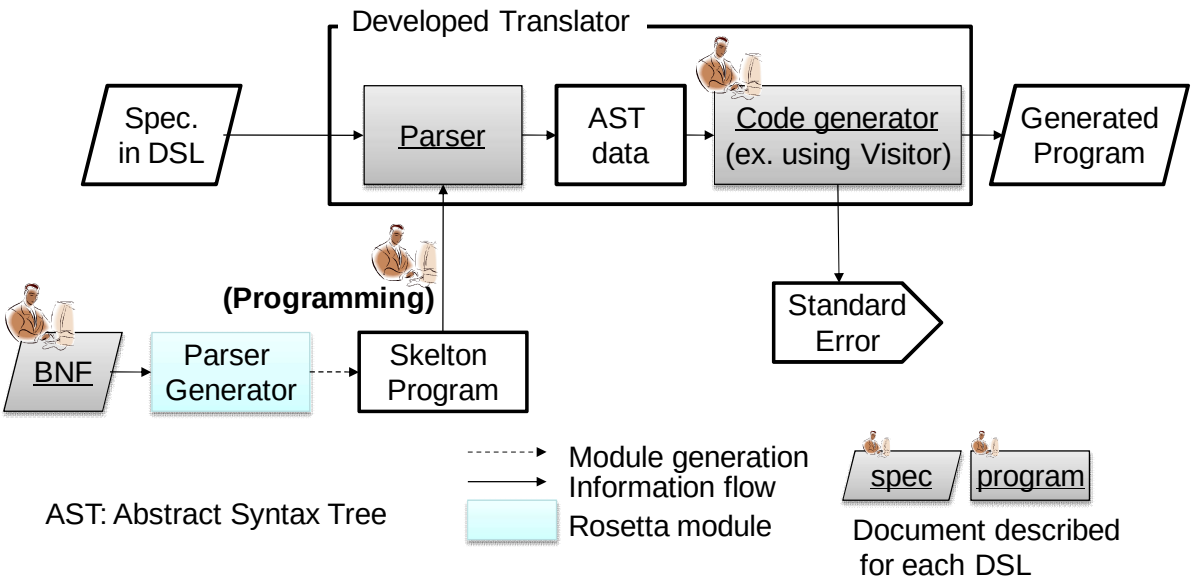

Figure 14: Conservative method for developing translator

4-3-2 Orion: Parser generator

Orion is a parser generator which Rosetta provides. The output of Orion is source codes of Parser Generator JavaCUP [82]. The code generated by Orion includes BNF rules expressing syntax of the DSL and actions of BNF rules. These actions are program codes generating an AST of the DSL. For example, consider the following is BNF rules of simple mathematical formula.

```
exp      ::= exp PLUS term
           |   term ;
term     ::= term MULTI factor
           |   factor;
factor ::= LPER exp RPER ;
           |   NUM ;
```

Non-tenrminal symbols are “exp”, “term”, and “factor”, terminal symbols are captal words. “PLUS” is “+”, “MULTI” is “*”, “LPER” is “(“, and RPER is “)”. “NUM” is a integer token. For the above BNF rules, we give specifications of Orion. It is called an **AST specification** as is shown in Figure 15.

```
exp ::= (left) exp (op) PLUS (right) term
      |   (exp)   term.factor.exp
      |   (num)   term.factor.NUM
      |   (term) term                ;
term ::= (left) term (op) MULTI
           (right) factor
      |    (exp) factor.exp
      |    (num) factor.NUM     ;
factor ::=   (exp) exp
           |    (num) NUM   ;
```

Figure 15: Example of AST specification

Terminal symbols and non-terminal symbols in the above represent nodes of a generated AST. The term left to “::=” is a parent node of nodes in rights. A **branch name** *name* is given in parenthesis “(*name*)” before each terminal or non-terminal sybmol in the right side of a rule. A branch name is given to a branch in the AST from the parent node in the

left of the rule to its child node of the symbol in the left. There is an expression like “factor.exp” in Rule “term”. It means that the child node with branch name “exp” of the parent node expressed in “term” is node expressed in “exp” followed by “factor”. In short, the “exp” node directly connected to the “term” node without “factor”. This mechanism symplifies an AST. The BNF sometimes include some non-terminal symbols for techincal reasons. They are necessary to express a complex syntax. They may be comfusing the DSL developers. The mechanism eliminates these techincal non-terminal symbols. Orion and JavaCUP generates a parser which generates an AST shown in Figure 16 when the parser reads Formula “3*(2+1)”. “term”, “left”, “op”, “right”, and so on are branch names.

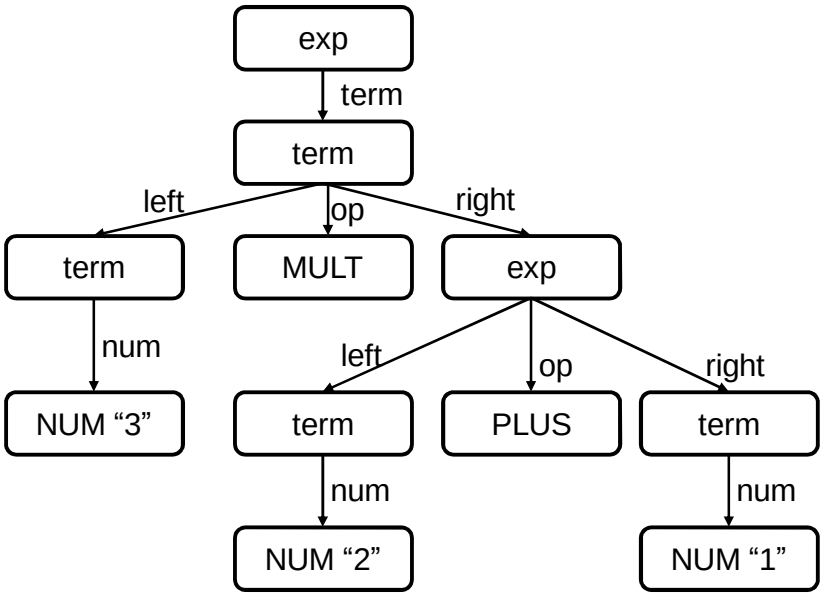

Figure 16: Example of Abstract Syntax Tree

4-3-3 Code generator and output template

The code generator accesses an AST, such as Figure 16, stored in the AST database, and generates program codes. The rule of the code generation is written in output templates. Figure 17 is an example of output templates for the AST in Figure 16. This example is written in File “calculator.gen”. File “exp.gen” included into “calculator.gen” with an INCLUDE tag in “calculator.gen” is shown in Figure 18. We omit, in this thesis, the explanation of “term.gen” included in “exp.gen”. The code generator with “calculator.gen”

generates program codes shown in Figure 19.

An output template consists of two parts like HTML:

- Text part: The code generator directly generates contents of text parts.
- Tag part: Tag is a part surrounded by "<" and ">". The code generator interprets the tag, and generates suitable codes corresponding to the semantic of tag. For example, when the code generator encounters an "INCLUDE" tag, it reads the file specified in the argument of the "INCLUDE" tag, and it interprets the file.

An output template similar to HTML syntax has following advantages:

- It is familiar because it is similar to the HTML syntax,
- The DSL developers can easily guess the generated code because of the text part,
- The output templates provide several strong data structures: a pointer to a node in AST (pointer type), a string type, and a stack type. These are useful for accessing an AST. The pointer type is a data type to indicate a node of AST. An example of a variable of a pointer type is "&EXP" in Figure 18. The stack type is an array of string types or point types.
- The output templates provide a node search expression. It can find a node in a specific subtree, and return a pointer type data. For example, Expression "$(&EXP/exp:*:)" returns a node under Branch "term", the node which under Branch "exp" in the subtree whose root is pointed by "&EXP".
- An INCLUDE tag enables subroutine calls. In addition, the code generator can generate program codes of recursive structure with an INCLUDE tag. For example, The INCLUDE tag in "exp.gen" shown in Figure 18 includes "exp.gen" itself.
- EXISTS tags and others enable conditional jumps. For example, in "<EXISTS $(&EXP/op:*:)>", if the subtree pointed by "&EXP" has a child node with branch "op", then the code generator interprets the region between "<THEN>" and "</THEN>".
- FOR tags enable a loop. The argument of a FOR tag is a stack data type. It may matches plural nodes in a specific subtree. The code generator repeatedly interprets the region between "<FOR>" and "</FOR>" until the argument of FOR matches nodes. An example of the argument of "FOR" is "(&EXP, /num:*:)". It finds all the nodes with branch "num" in the subtree pointed with "&EXP".

```
/* Print out the evaluation result. */
```

```
#include <$lt>stdio.h<$gt><p>

int calc() {<p>
 int ans = 0;<p>
 ans =
<INCLUDE "exp.gen">
    <POINT name=EXP, address=&(/ROOT:*:)>
</INCLUDE>;<p>
 return ans;<p>
}<p>

void main(int argc, char* argv[]) {<p>
 printf("Answer = %d¥n", calc());<p>
}<p>
```

Figure 17: Example of output template: “calculator.gen”

```
<EXISTS $(&EXP/op:*:)><THEN>
  <INCLUDE "exp.gen">
   <POINT name=EXP, address $(&EXP/left:*:)>
  </INCLUDE><s>+<s>
  <INCLUE "exp.gen">
    <POINT name=EXP, address $(&EXP/right:*:)>
  </INCLUDE>
</THEN></EXISTS>
<EXISTS $(&EXP/exp:*:)><THEN>
 (<INCLUE "exp.gen">
    <POINT name=EXP, address $(&EXP/exp:*:)>
  </INCLUDE>)
</THEN></EXISTS>
<EXISTS $(&EXP/term:*:)><THEN>
  <INCLUE "term.gen">
   <POINT name=TERM, address $(&EXP/term:*:)>
```

```
    </INCLUDE>
  </THEN></EXISTS>
  <EXISTS $(&EXP/num:*:)><THEN>
     <$(&EXP/num:*:)>
  </THEN></EXISTS>
```

Figure 18: Example of output template: "exp.gen"

```
  /* Print out the evaluation result. */
  #include <stdio.h>
  int calc() {
   int ans = 0;
   ans = 3 + ( 2 + 1 );
   return ans;
  }
  void main(int argc, char* argv[]) {
   printf("Answer = %d¥n", calc());
  }
```

Figure 19: Generated program codes

4-3-4 Development methodology in Rosetta

The development method in Rosetta is shown in Figure 20. We assume that the DSL developers are a designer of a DSL, a designer of the DSL translator, a developer of the translator. Numbers in Figure 20 corresponds to the following steps:

1. A designer of the DSL, who has much domain knowledge, defines the syntax and semantics of the DSL, by using his knowledge and experience. The syntax is defined in BNF, and the semantics is specified in his natural language.
2. A designer of translator defines specification of generated code. It specifies the output of the DSL translator. Example specifications of generated code shown in Figure 19 are shown in Figure 21. "[…]" in the figure shows one of conditional branches. One of the branches is selected if a comment referred by number in its right shoulder is satisfied. "<…>" in the figure shows a loop. A comment referred by number in its right shoulder shows the condition that the loop continues. A Developer of translator refers to the

specification

3. The designer of the DSL translator specifies an AST in the AST specification like Figure 15.

4. The designer of the DSL translator specifies specifications of semantic error check items. An item specifies a semantically incorrect condition, and an error message for that condition.

5. The designer of the translator defines specification of command line including command syntax and its semantics of the DSL translator.

6. The developer of the translator gives Orion a pair of BNF and specification of AST. Orion with JavaCUP generates a parser program codes.

7. The developer of the translator makes a main routine program of the DSL translator by referring the specification of command line.

8. The developer of the translator specifies output templates of the DSL translator, by referring the specification of generated codes and an AST specification.

9. The developer of the translator makes a program of the semantic checker by referring the specification of semantic error check items

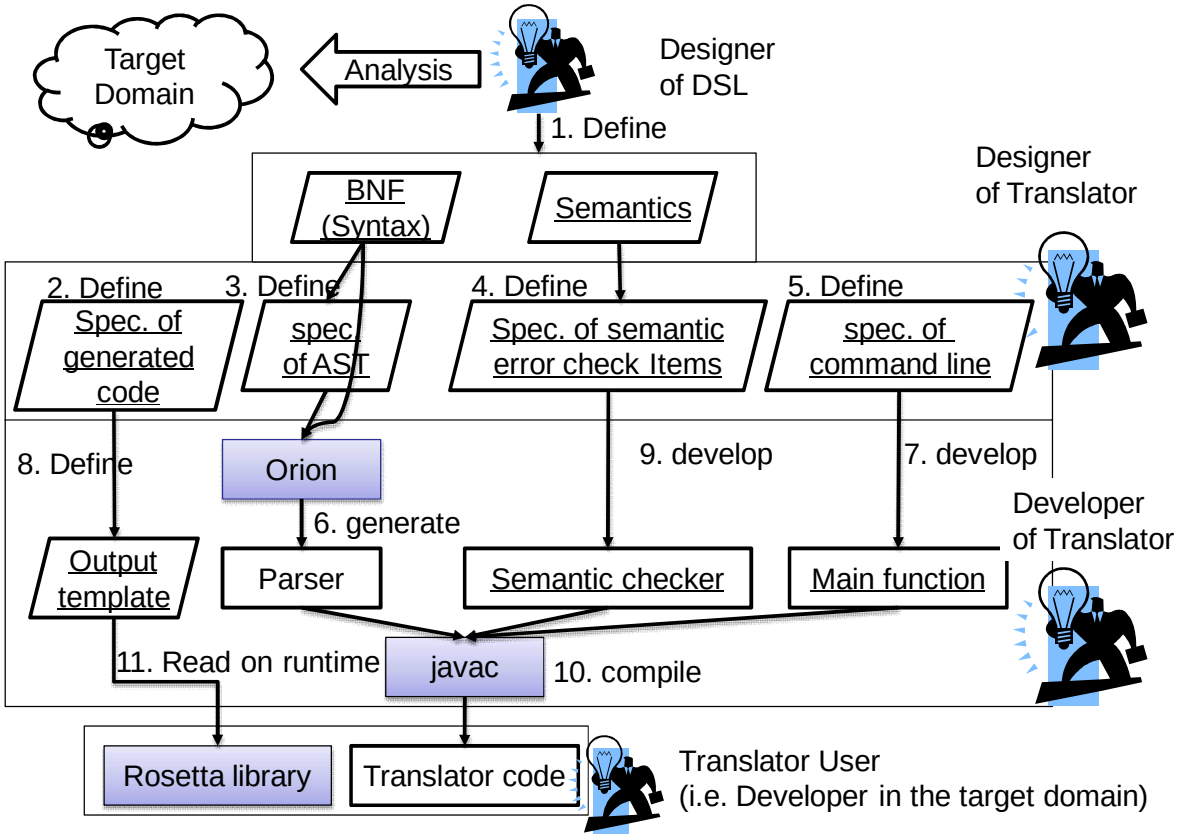

Figure 20: Rosetta development methodology

```
/* Print out the evaluation result. */
int calc() {
    int ans = 0;
    ans =
[       < exp>*5 + < exp>*5       ]*1
[
  [     <term>*6 *   [( <exp>*5})]*4    [ num*6 ]*5 ]*3
  [     <exp>*5   ]*7
  [     num*6   ]*8
]*2
[       num*6     ]*3
[       exp*5     ]*4;
```

```
    return ans;
}
void main(int argc, char* argv[]) {
    printf("Answer = %d¥¥n", calc());
}
```

1. Case when branch *op* is under Branch *exp*.
2. Case when branch *term* is under Branch *exp*.
3. Case when branch *num* is under Branch *exp*.
4. Case when branch *exp* is under Branch *exp*.
5. Branch *exp* is recursively called

:

Figure 21: Specification of generated code

4-3-5 Improvement of maintainability in Rosetta

The following advantages of Rosetta improve the maintenability of a DSL and its translator.

- Orion generates a parser program, according to a specification of AST. The generated AST is always in accordance with the specification of AST. In short, the specification of AST is a reliable design document. There are often the cases where design documents are not in accordance with the program because the design documents are not updated even when program codes are updated. However, by using Orion, we can avoid this situation.
- A DSL translator generates program codes similar to its output templates. The DSL developers can easily find correspondence between the generated program codes and the output specifications. For example, Line 1 of the generated code shown in Figure 19 corresponds to Line 1 of Output template “calculate.gen” shown in Figure 17. The DSL translator developers, therefore, easily find where to change the output templates when generated codes must be changed. In addition, when they changed the output templates, they easily guess the impact on the generated codes.
- The specifications of the generated codes can be specified into several outpute templates files. The DSL translator developers can run the translator with a part of the output templates. It is useful for debugging and test of the DSL translator.

- The output templates have features strong enough to access ASTs. The syntax of a DSL is context free grammer, and it is defined in BNF. The BNF has a conditional branch ("|") and a recursive operation with reference of a non-terminal symbol in the right of a rule. The output templates have features of a conditional branch and a recursive operation, too. The output templates also have a feature to call a Java library. We can use, for example, calcurations, which are not supproted in the outpute templates, by using a Java library.

4-4 GDMO translator: An Example of Rosetta

We developed the Q3 agent platform in Chapter 3. The development cost of Q3 agent application program on the platform is also a critical issue. In Q3 interface, the management model of each network element is given in Guidelines of Definition of Managed Object (GDMO) standardized in ITU-T [12]. The GDMO is a kind of DSLs for a Q3 agent. We developed, by using Rosetta, a DSL translator. It reads GDMO templates, which are specifications of an NE written in GDMO, and some original specifications, and generates programming codes running on the Q3 agent platform, as shown in Figure 22. We call the translator a GDMO translator. The behavior of managed objects are specified in BEHAVIOUR templates in a natural language. The GDMO translator cannot handle the natural language, and it cannot generate program codes of the behavior of managed objects. In stead, the DSL developpers write the program codes.

A Q3 agent application program includes about 100 managed object classes and initialzation codes of several ten thousands of managed object instances. The specifications of GDMO and generated code tend to be large. For example, we give the GDMO translator totally 17kL of GDMO templates and other specifications about managed object instances, and it generates 221kL C++ program codes. This means that the application developer skip writing 221kL. The almost parts of GDMO templates are standarized such as in M3100.

The application developers just use sutable set of templates and write some of additional specifications. Therefore, the GDMO translator save the development cost of Q3 agenet application program. We can update a Q3 agent application program by changing the specifications, and by re-generating program codes with the GDMO translator. For example, the GDMO translator reads an specification of a containment tree, which containes managed object instances in a tree structure. The Q3 agent application developers can

modify the specification, and they easily change the positions of managed object instances in the containment tree.

The Q3 agent platform is updated 23 times and 80-times maintenances as a result. These maintenance is classifiled as follows:

- **Maintenance for repairing** occurred 60 cases,
- **Maintenance for adaptation** occurred 2 cases,
- **Maintenance for perfectness** occurred 12 cases, and
- **Maintenance for affected modules** occurred 6 cases.

About 8% of the maintanences are maintenances for affected modules. The GDMO translator must be maintined in these cases.

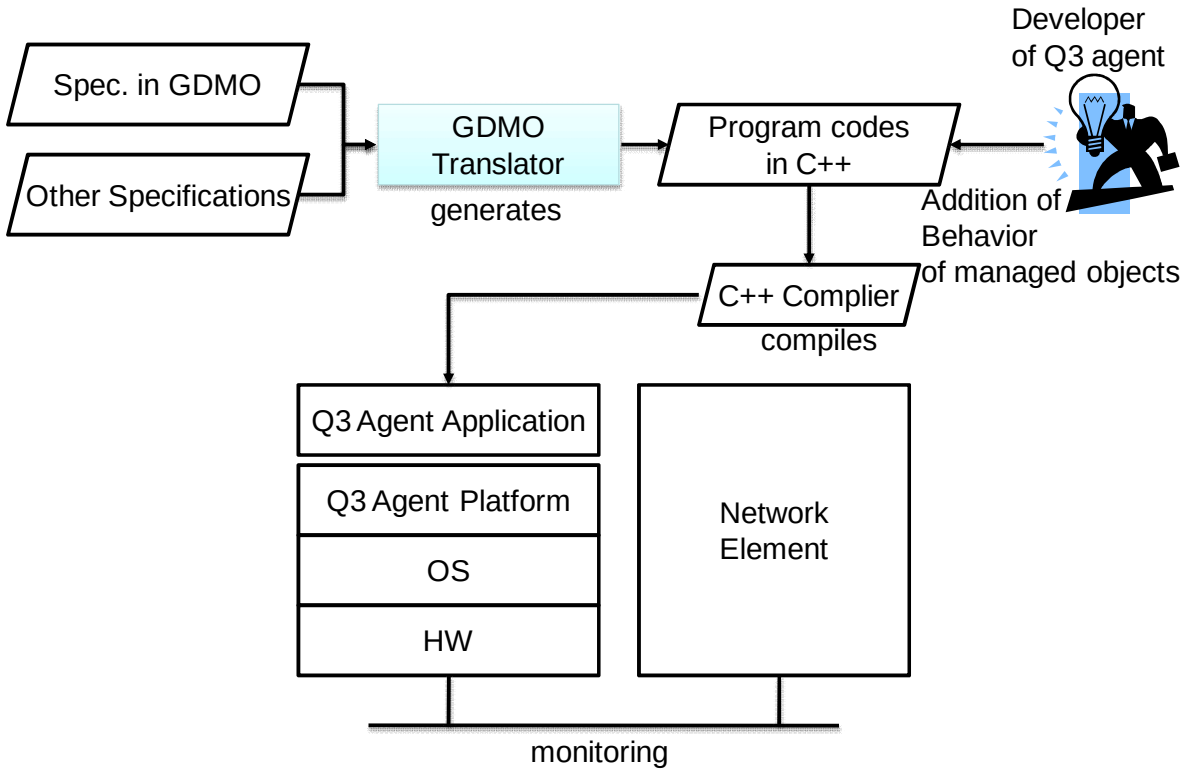

Figure 22: GDMO translator and Q3 agent platform

4-5 Evaluation with experiment and discussion

4-5-1 Measurement of maintenance costs

We compared development cost and maintenance cost of a DSL with Rosetta with those with a visitor design pattern [63]. The visitor pattern is suitable for a translator program because each method in a visitor object is invoked when the object encounters a certain type of an AST node. We developed two HTML generators with Rosetta and with visitor pattern. Both the HTML generators read GDMO templates and definition of data in ASN.1, and generates HTML files which express definitions in the GDMO templates and the ASN.1 data. Each of two programmers developed each of the HTML generators. They have experiences of development projects with Java, but they had not have experience of usage of Rosetta before.

In this experiment, we developed only code generation parts. We used a parser of the GDMO translator. We defined the specification of generated codes and the specification of the AST of the GDMO translator. The two programmers refer these specifications, and develop the HTML generators.

Firstly, the two programmers developed the first version of the HTML generators, and we measured man powers used for these developments. After these developments, we give the two programmers specifications of the second version of the HTML generators. They develop the second versions, and we measured this maintenance man powers.

Table 10 shows the man powers for these developments. "# of lines" is the number of lines of the GDMO translator (ver.1). "# of modified lines" is the number of lines modified in the second version from the first version. The number of the modified lines is measured with Command "diff" in UNIX.

It shows that the maintenance cost with Rosetta is smaller than that with the visitor patterns. However, the number of modified lines with Rosetta is larger than that with the visitor pattern. The total number of updated points with Rosetta in the second version is less than that with the visitor pattern. It means that modified points are distributed in the development with the visitor pattern. It is also shown in "#modified lines per modified points" in Table 10. The modified parts are inherently distributed in the visitor pattern because each visitor method corresponds to a node of AST. The output program codes are generated from the information of several AST nodes. Therefore, they are generated by several visitor methods corresponding to the several AST nodes. Maintenance is difficult if

the points to be modified are distributed in the translator program.

Additions to program codes in the development with Rosetta are more than that with the visitor pattern (Table 11). If new functions are added to the HTML generator, we just add codes for the new functions into output templates of Rosetta. In the development with the visitor pattern, some of existing visitor methods must be modified. It is a reason why code modifications in development with the visitor pattern are more than those with Rosetta. Code additions are generally easier than code modifications because code modifications often have impacts on the existing codes. A programmer must take care that modifications is independent from existing codes.

Table 12 shows the man power for repairing bugs for a trouble ticket. It is smaller in development with Rosetta than that in the visitor pattern.

Table 10: Cost of Development and Maintenance

	Ver. 1 (development)		Ver. 2 (maintenance)		
	Man power [person hour]	# of lines (kL)	Man power [person hour]	# of modified lines (kL)	# of modified lines (L) per modified points
Rosetta	200	9.0	53	4.0	11
Visitor	192	8.0	87	2.3	6.3

Table 11: Updated points in the second version

	Addition	Removal	Modification	Total
Rosetta	211	17	123	351
Visitor	88	88	192	368

Table 12: Cost of repairing bug in HTML generator [person hour]

	Ver.1	Ver. 2	Average
Rosetta	2.39	2.11	2.45
Visitor	3.04	2.68	2.77

4-5-2 Performance

DSL translators developed with Rosetta are slow. The GDMO translator takes 5 – 6 hours for 20kL inputs. It also takes 100MB heap memory. It is because the Rosetta generates a

large AST of the whole inputs. It takes a lot of heap memory. In addition, searching the large AST for nodes takes a lot of time.

Alternative method is a streaming processing like SAX [83]. SAX is one of the XML parsing methods. DOM, which is another XML parsing method, generates an AST, just like Rosetta. The DOM, therefore, takes a lot of heap memory as well as Rosetta does. The SAX sequentially reads an XML file and generates program codes. This may be a hint for us to improve the performance of Rosetta. However, the code generation which requires some information distributed in an AST is not suitable to the SAX-like method. The code generation with the SAX-like method can only use local information where the SAX is now reading. Because information of a managed object class is distributes in nine kinds of GDMO templates, the GDMO translator does not suitable to the SAX-like method.

4-6 Conclusion

The development cost and the maintenance cost of Q3 agent applications are keys for reduction of capital costs of operation system. There are several approaches with DSLs for reduction of application development costs.

We pointed out the maintenance for affected modules must be considered in DSL approach. We clarify that about 8 percents of maintenance is the affected maintenance through our Q3 agent development project. We proposed a DSL toolkit called Rosetta and its development method in order to reduce the development costs and the maintenance costs including the affected maintenance. The specification of AST is a reliable design document of AST because Orion generates a parser program from the specification. The output templates are easy for the developer to maintain the DSL translator. It reduces the maintenance cost. We compare the proposed method with the visitor pattern, and proved the advantage of the proposed method through our experiments.

Chapter 5

Low cost operation for high-dependable systems

– Fault analysis for implicit failure derivation –

Chapter 5. Low cost operation for high-dependable systems -- Fault analysis for potential failure derivation --

5-1 Introduction: Issue of low cost operations

System management consists of Fault Management, Configure Management, Account Management, Performance Management, and Security Management, which are called FCAPS. Management consists of four steps (PDCA): Plan a management purpose, Do an operation of the managed systems with monitoring the situation of the operation systems, Check the systems whether a problem occurs or not, and Action a problem repair. Administrators are deployed to do this PDCA cycle. This results in expensive personnel costs. Especially in fault management, the expertise knowledge is required to detect anomaly and to analyze the cause. However, the expert persons require higher personnel costs than normal system administrators. We focus on the fault management in this thesis. Especially, we try to help expert to find the cause of fault.

5-2 Classification of faults

We firstly define the terminologies in fault management. Figure 23 shows terminologies. **Defect** is a cause, which may cause performance degradations and malfunctions. **Fault** is the situation where the managed system causes performance degradations and malfunctions, but administrators or users are not aware it. **Failure** is performance degradation or a malfunction of managed systems explicit to the administrators and the users. **Incident** is an impact, caused by failures, on user or society. For example, a program without freeing the used memory space may cause a memory leak. This is an example of a defect. This causes a memory leak. This is a fault. This memory leak consumes a heap memory area, and it causes the slashing. This results in performance degradation of response time of the managed system. This is an example of failure. Users cannot do his job before their dead lines because of this performance degradation. This is an example of incident.

We define other terminologies. An **event** is a notification or a record of the situation of managed system. It may be issued or recorded in to a log when the situation of managed system is changed: for example, it may be issued when configuration of the system is changed, when a fault is found, or when a request comes to the managed system **Anomaly** is a situation where the system is out of normal situation. It may be a fault or it may be caused by other causes, such as security attack. **Failure derivation** is a situation where a failure causes some other failures. The caused failures are called **derivative failures**. An example of failure derivations is a response time degradation on an AP server caused by a

performance degradation of a DB server which the AP server access. **Root failure** is a root failure of a failure derivation.

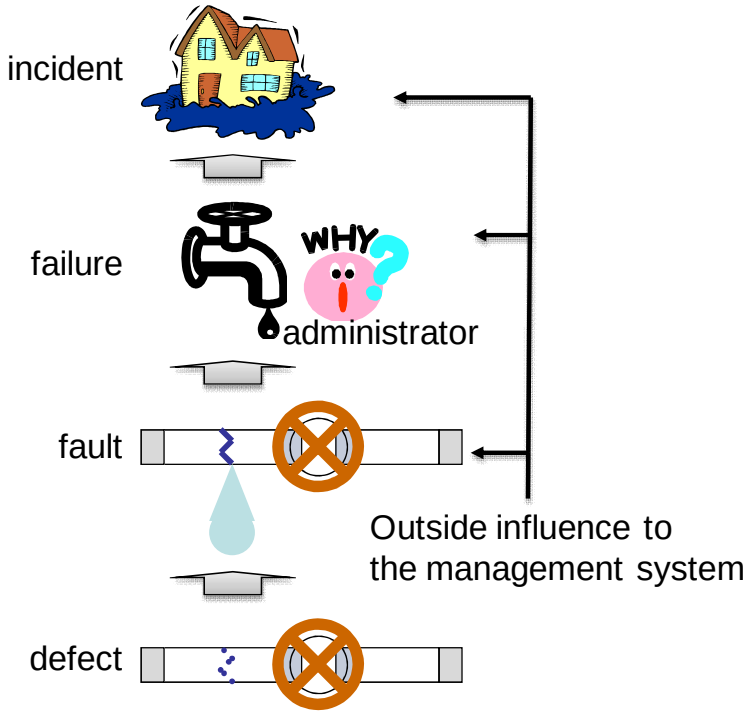

Figure 23: Terminologies in Fault Management

Figure 24 is a fault model. A managed system consists of some subsystems which depend on each others. Some subsystems may have defects in their selves. Influence of environment around the managed system may trigger failures caused from the defects. Some failures are derived on another subsystem depending on the subsystem with the failures. The managed system may issue the events of these failures.

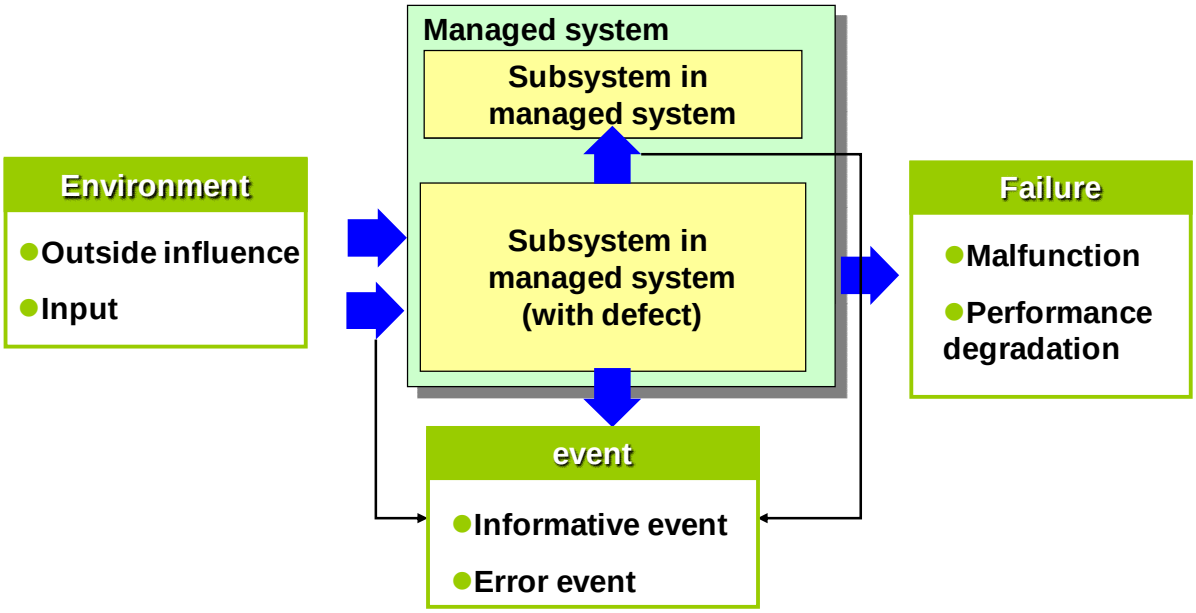

Figure 24: Fault model

We classify failures on a map shown in Figure 25. Firstly, we classify failures into **known failures** and **unknown failures**. The former has occurred beforehand and the administrator has knowledge about it. For example, some hardware failures occur in certain rate. The hardware failures occur usually in a large managed system including many hardware devices. Another example is a software bug with memory leak. The administrator knows the symptoms and the resetting the system fixes temporarily the problem. The administrator should fix a bug, but finding a bug is difficult. He, therefore, resets the system and temporary repairs the system. The known failure usually occurs and, therefore, the management cost of known failures is dominant.

The latter has never occurred beforehand, and the administrator has no knowledge about it. It scarcely occurs, but it takes a long time for fixing a failure when it occurs.

We classify failures with another view into silent failures and failures with events. The managed system or the operation system of the managed system tries to find an anomaly, to issues events to notify the administrators of the anomaly. These failures are classified into the failures with events.

The managed system cannot issue events even in anomalous situation. For example, the managed system goes down immediately and it does not have time to issue an event.

The failure without event is called a silent failure.

We classified the failures in the view of failure derivations. There are five classes in the view of failure derivations: **none** means that no derivative failure occurs. **Explicit** derivation of failures means that a failure derivation relation is estimated from the structure of managed systems. For example, equipments connected to a router may cause connection errors when the router causes a failure. This derivation relation is estimated from the relation of physical links between the routers and equipments. **Implicit** derivation of failures means that a failure derivation relation is difficult to be estimated. For example, failure derivation in application programs is usually complex and is difficult to be estimated because the relation between software components are complex and the application program is usually a black box to administrators. Derivation relation is **dynamic** when the relation changes dynamically. For example, deployment of application programs running on virtual machines may change. This virtualization changes the relation between application programs and machines where they run. **External** means that environment around the managed systems causes a failure. For example, a thunderbolt causes a surge voltage on the power lines of the managed systems, and it results in a failure.

The fault management methods are classified into the following three:

- **Anomaly detection** is to find that an anomaly happened in the managed systems. It does not matter what anomaly is and where it happens.
- **Fault localization** is to find where an anomaly happens.
- **Root cause analysis** is to find where an anomaly happens and to find why the failure causes.

We think that there is no almighty method which can analyze the cause of all types of the failures. Each problem in Figure 25 requires each method. We propose a failure derivation analysis method for failure with events in this section. This method covers the area shown in Figure 25.

The characteristics of proposed method are the following:

- The method can learn derivation relation model from event log including failures. It can find a root cause of failures with events from a lot of events, considering the learnt derivation model.
- It can find a root cause of failures in a distributed system, while their configurations are different from each other. The derivation relations are different from each other based

on their configuration. For example, a web three-tier system without a load balancer becomes insufficient when an application server goes down. A web three-tier system with a load balancer can continue its services even if one of application servers goes down. It is, therefore, difficult to find derivation relation because the derivation relation depends on the configuration of the distributed systems.

- It can find a root cause of failures with implicit failures. Failure derivation relation is implicit when a fault occurs in application software. The structure in software is complex and it is a black box to administrator. The system can analyze the implicit derivation in software as well as in the system level.

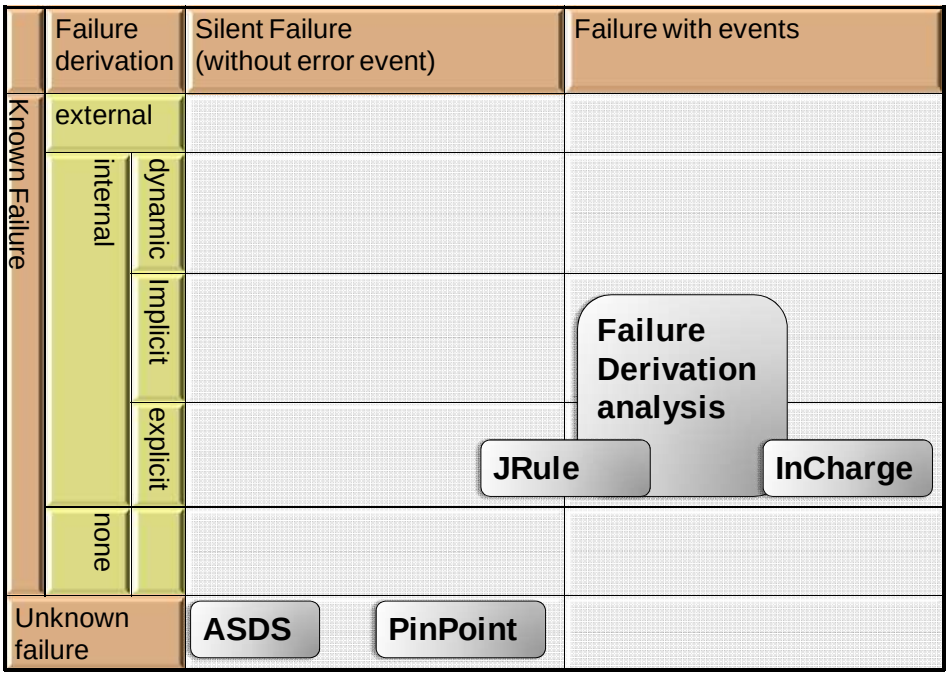

Figure 25: Classification of failures

5-3 Related work

There are a lot of event correlation methods, which correlate a lot of events with each others. It correlates, into an event group, a lot of events which are caused by a fault. An administrator can easily understand the situation with an event into which a lot of events are correlated. Many works of event correlation use rules expressing which events are

related [84]. Rules are used for root cause analysis as well as event correlation [85][86]. However, these rule-based analyses are difficult to be applied to a fault in an application program. An application program is a black box to an administrator, and therefore, he cannot write down the rules. Even if he knows the details of the application program, the relation among modules in the application program is too difficult to be expressed in the rules.

The rule-based approach is also difficult to be applied to a distributed system because the failure derivations are different depending on the configuration of the distributed systems. For example, the failure derivations are different between a three-tier system with a load balancer and that without a load balancer. Clients can use the service even when one of application servers connected to the load balancer is down. However, the clients cannot use the service in the former system when the application server is down. We have to prepare different rules for these two systems.

Mean Time to Recovery (MTTR) is conserved important than Mean Time to Failure (MTTF) in Recovery-oriented System (RoC) project [88]. This is because first recovery is essential to availability of systems. MTTF only tells that the system is hard to be failed, but it does not tell how long it is in failed situation. It is important for short MTTR that an administrator can find where a fault occurs in a short time. PinPoint [89] is a fault localization tool developed in RoC project. It monitors elapsed times in a control flow of a J2EE application with probes injected in the J2EE application. It can compare the elapsed time in failure condition with that in normal conditions. It can be a fault point that the elapsed time is much longer than normal elapsed times. PinPoint requires the probe injections, and a lot of systems cannot admit the injections.

In InCharge [90], an administrator provides a “code-book” matrix. It represents relations between events and faults. In short, each row of the code-book matrix $\mathbf{C}$ represents a fault, and each column represents an event. Element (i, j) of Matrix $\mathbf{C}$ is 1 if Fault i occurs and Event j is issued. It is 0, otherwise. InCharge uses the code-book matrix, and analyzes the root cause of a failure. The managed system issued a set of events when a failure occurs. InCharge expresses the set as Vector $\mathbf{e}$ whose element j is 1 if Event j is in the set and 0 otherwise. Then, Fault i is expressed in Element i of Vector $\mathbf{Ce}$. This means that the hamming distance of Vector $\mathbf{e}$ is most similar to Row i of Matrix $\mathbf{C}$ in the view of humming distance. InCharge can identify the fault firstly, but it requires the

administrator to define the code-book. He must write information about the events and the managed system in Managed Object DEfinition Language (MODEL). 1,170 lines are required for 40 managed classes in the evaluation in [90]. In addition, a code-book is a kind of rule. Therefore, the administrator cannot define a code-book for a managed system with implicit failure derivation.

Smart Shifter [91] is an intrusion detection system, and it can detect abnormal operations to the managed system. It models a sequence of operations to the managed systems as the hidden Markov model. In short, it learns normal sequences of the operations, and then it finds an outlier in comparison with the learned sequences. The algorithm may be easily used in fault analysis as well as in the intrusion detection. It can detect an anomaly which may be a failure, but it cannot find a root cause of it.

ASDS [92] finds invariants between performance parameters in a managed system. It can detect an anomaly when some of invariants are broken. It is an anomaly detection system, but it is not a root cause analysis system.

5-4 A Solution for implicit failure derivation

5-4-1 Overview

We propose a method which reads exiting logs of events, estimates a failure derivation model, and analysis the root cause of failure by using the estimated model. It can analyze the implicit failure derivations because it can estimate the model from the existing logs. We assume that the logs are outcomes of implicit derivation.

This method can handle faults in Area “Failure derivation Analysis” in Figure 25. It can handle a fault with events. The fault can cause the failure derivation, and the method can solve the implicit derivation. It can also solve the explicit derivation, but rule-based approach, such as JRule and InCharge, may analyze the root cause more precisely. The rule which an expert administrator defines is generally more informative than the derivation model which the system estimated from exiting logs.

We show the steps of the proposed method in Figure 26. “Error Event Specifications” in Figure 26 is a list of events the managed system issues. Firstly, the failure derivation analysis system reads it and generates an initial failure derivation model (Step 1). The initial failure derivation model is an initial state of the iterative algorithm mentioned later.

Secondly, the analysis system reads the exiting log of events for learning. The analysis

system estimates the failure derivation model (Step 2). It generates the failure derivation model from the initial derivation model with the iterative algorithm. The SI-er can use the log issued in the test phase of the managed systems.

Finally, the analysis system is applied to the managed system running. It analyzes the log which the running system issues, and guesses the root causes of events in the log, by referring the tuned failure derivation model.

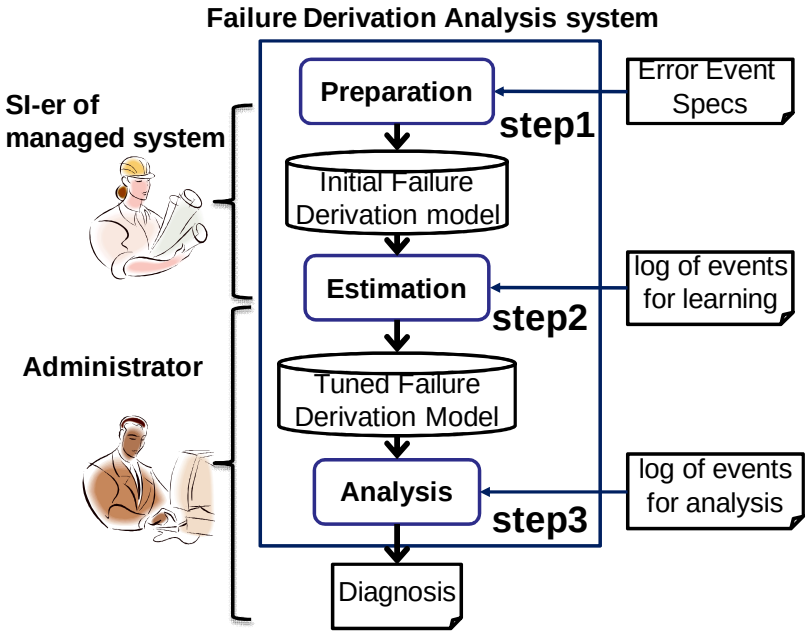

Figure 26: Proposed Method

We give an example of the event specifications in Figure 27. The event specifications consist of Paragraph "[events]" and Paragraph "[states]". The former defines events the managed system issue. The latter specifies the failures which are causes of the events. In Paragraph "[events]", we can define an event with regular expressions. For example, the following definition of event includes "(¥d+)", which expresses an arbitrary process id.

"`child process (¥d+) still did not exit, sending a SIGTERM`"

In Paragraph [event], we define a failure which causes the event. The following example means that Failure "httpd is stopped" issues Event "child process …".

"`child process (¥d+) still did not exit, sending a SIGTERM`", "`httpd is stopped`"

```
[states] # "Faiures"
"httpd is stopped"
"JBoss is stopped"
"MySQL is stopped"
"httpd NW is stopped"
"JBoss NW is stopped"
"MySQL NW is stopped"
"httpd started"
#(ommition)
[events] # "event","failure"
"httpd(起動|startup)
succeeded","httpd started"
"httpd(停止|shutdown) succeeded",
```

Figure 27: Example of Event Specifications

We show the GUI of the prototype of the failure derivation analysis system in Figure 28. The file names and parameters used in the system are placed in the upper part of the GUI. The analysis result is shown in the lower part. The result is shown in the tree view in the left of the lower part, and the same information in text is shown in the right of the lower part. The top level of the tree view are called event regions, which are mentioned later. A subnode of an event region includes a root failure and the derived failures. The first node under the subnode expresses the root failure and the following in the subnode is derivative failures.

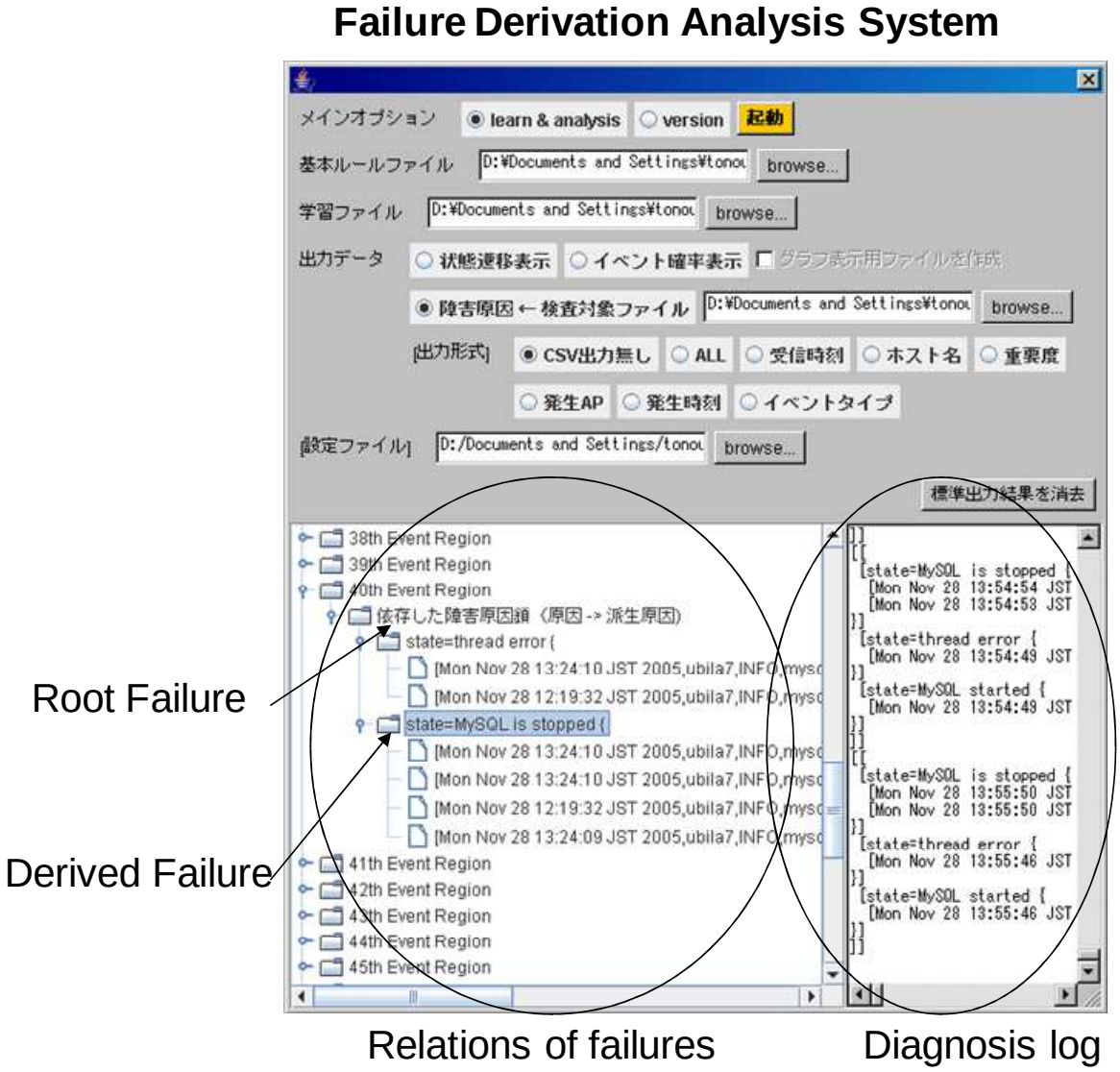

Figure 28: Prototype system

5-4-2 Approach

We assume the following assumptions.

- Events are issued in the same order with the derivation of the failures. Failure *A* issues Event e_A, and Failure *B* issues Event e_B. Failure *B* is derived from Failure *A*. Then, Event e_A is issued before Event e_B.
- Probability that Failure B occurs under Failure A is independent from other Failures.

We think that the former assumption is valid when the managed system is monitored in a short period. In short, Failure B occurs after Failure A because of the causality of failure derivation. The event is almost instantly issued if the monitoring period is short. Therefore, Event e_B is issued after Event e_A. However, we must pay attention to the following exceptional cases. If the monitoring period is not short, these two events may have the same time stamp. Components in the managed systems have the same clock with Network Time Protocol (NTP), but they have time-errors among them.

We mention the latter assumption. Some failures may be derived from plural failures. A failure may occur when both the active system and the stand-by system in a redundant system are out of order. We, however, assume that a lot of failures are derived from one failure.

We make a model of failure derivations as Hidden Markov Model (HMM). HMM M is consists of $M = \{S, \Sigma, A, E, Pr_0(S)\}$. $\Sigma = \{\sigma_0, \ldots, \sigma_{n-1}\}$ is a set of observal events. $S = \{s_0, s_1, \ldots, s_{n-1}\}$ is a set of hidden states, which issue a set of the observable events. A is a transition probability matrix, which expresses the transitions between the hidden states. E is a matrix whose element is a probability that an event is issued under a failure. $Pr_0(S)$ is probability of the initial state.

We assume that an event is regarded as an observable event in HMM, and a failure is regarded as a hidden state. The following describes this modeling in detail. In the following, we use $\Pr(a \mid b)$ as probability of a under b.

- A set of failures in the managed system is defined as $S = \{s_0, s_1, \ldots, s_{n-1}\}$. Failures are defined in Paragraph "[states]" in the event specifications mentioned before. s_0 is a normal state.
- A set of events issued in the managed system is defined as $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{m-1}\}$. The set of events are defined in Paragraph "[events]" in the event specifications.

- Probability Matrix of events is $\mathbf{E} = \begin{pmatrix} \vdots & \\ \cdots & \Pr(\sigma_j \mid s_i) \quad \cdots \\ \vdots & \end{pmatrix}$. Probability $\Pr(\sigma_j \mid s_i)$ is Element (*i, j*) of E, and it is a probability of Event σ_j under Failure s_i.

- Probability Matrix of failure derivation is $\mathbf{A} = \begin{pmatrix} \vdots & \\ \cdots & \Pr(s_j \mid s_i) \quad \cdots \\ \vdots & \end{pmatrix}$. Probability $\Pr(s_j \mid s_i)$ is Element (*i, j*) of A, and it is a probability of Event s_j under Failure s_i.

- Vector $\Pr_0(S) = (\cdots \quad \Pr_0(s_i) \quad \cdots)^{\mathrm{T}}$ is a set of probability that the initial state is Failure s_i. M^{T} means a transport matrix of Matrix M.

  We define the values of $\Pr_0(S)$ as the follows: $\Pr_0(s_0) = 1$ and $\Pr_0(s_i) = 0$ if $i \neq 0$ because the system should begin with the normal state s_0.

Figure 29 shows an intuitive image of the model. The bottom part of the rectangular in the figure shows the probability matrix of failure derivations. For example, the sick lines in the bottom part show a sequence of transitions that Failure “DB Process Down” causes Failure “DB Connection Down”, and Failure “DB Connection Down” causes Failure AP Server Busy”.

The upper part of the rectangular shows a set of the events $\Sigma = \{\sigma_0, \sigma_1, ..., \sigma_{m-1}\}$. A thin line in the bottom part shows that, for example, Event “mysql ended” is issued from Failure “DB Process Down”.

We described Step 2 in the proposed method. Assume that a sequence of events $\overline{\sigma^l} = \sigma^l{}_0, \sigma^l{}_1, ..., \sigma^l{}_{m-1}$ is given as a log of events for learning. “*l*” means “learning” and, of

course, $\sigma_i^l \in \Sigma$. We calculate Probability Matrix of Failure Derivation A and Probability Matrix of Event E. In short, we calculate these two where the probability of occurrence of $\sigma_i^l \in \Sigma$ should be maxmum. These are given in $(\mathbf{A},\mathbf{E}) = \underset{\mathbf{A},\mathbf{E}}{\operatorname{argmax}} \Pr(\overline{\sigma^l} \mid \mathbf{x_0}, \mathbf{A}, \mathbf{E})$. Baum-Welch algorithm [87] can solve this equation approximately. We briefly explain the algorism. We can calculate the probabilities of occurrences of Sequence $\sigma^l{}_0,...,\sigma^l{}_{k-1}$ and Sequence $\sigma^l{}_k,...,\sigma^l{}_{m-1}$ in $\overline{\sigma^l} = \sigma^l{}_0, \sigma^l{}_1,..., \sigma^l{}_{m-1}$, where $\mathbf{A}$, $\mathbf{E}$ and $\Pr_0(S)$ are given. The algorisms of these calculations are known as a forward algorithm and a backward algorithm. We use these algorithms, and calculate approximate and temporal solutions of $\Pr(s_j \mid s_i)$ and $\Pr(s_j \mid s_i)$, where $\overline{\sigma^l} = \sigma^l{}_0, \sigma^l{}_1,..., \sigma^l{}_{m-1}$ are given and $\mathbf{A}$, $\mathbf{E}$ and $\Pr_0(S)$ are given. These are temporal solutions, and we describes these as $\Pr'(s_j \mid s_i)$ and $\Pr'(s_j \mid s_i)$. By using these temporal solutions, we can estimate $\Pr(s_j \mid s_i)$ and $\Pr(e_k \mid s_i)$ in the following equations.

- $\Pr(s_j|s_i) = \frac{\Pr'(s_j|s_i)}{\sum_{j'} \Pr'(s_{j'}|s_i)}$
- $\Pr(e_k|s_i) = \frac{\Pr'(e_k|s_i)}{\sum_{k'} \Pr'(e_{k'}|s_i)}$

We use these $\Pr(s_j \mid s_i)$ and $\Pr(e_k \mid s_i)$ as new elements of A and E, and we iteratively calculates these as $\Pr(s_j \mid s_i)$ and $\Pr(e_k \mid s_i)$.

Baum-Welch algorithm is an iterative algorithm, and it requires initial values of $\mathbf{A}$, $\mathbf{E}$ and $\Pr_0(S)$. We give an assumption that the failure derivation relations are implicit. We, therefore, assume that the probabilities in $\mathbf{A}$ and $\mathbf{E}$ are equal as initial values.

Initial values of elements of Probability Matrix A, for example, is given in $\Pr(s_j|s_i) = \frac{1}{n}$ where n is the number of elements in S.

Consider the initial values of Probability matrix of events $\mathbf{E}$. We define, in Paragraph "[events]" in the event specifications, the relation that a failure can issue some of events. We define n_i is the number of events which are issued by Failuress_i. We give the initial value of Probability $\Pr(e_k \mid s_i)$ is defined as follows:

- $\Pr(e_k \mid s_i) = \frac{1}{n_i}$, if s_i derives e_k, or
- $\Pr(e_k \mid s_i) = 0$, otherwise.

We consider Step 3. A sequence of events $\overline{\sigma^a} = \sigma^a{}_0, \sigma^a{}_1, ..., \sigma^a{}_{m-1}$ is given for a log of events for analysis. "a" in the left shoulder means "analysis". Of course it is $\sigma^a_i \in \Sigma$. We calculate, using $\mathbf{A}$ and $\mathbf{E}$ calculated in Step 2, a sequence of failures $\overline{s^a} = s^a{}_0, s^a{}_1, ..., s^a{}_{n-1}$ where the probability of the occurrence of $\overline{\sigma^a} = \sigma^a{}_0, \sigma^a{}_1, ..., \sigma^a{}_{m-1}$ is max. In short, we solve the following equation. $\bar{s} = \underset{\bar{s}}{\operatorname{argmax}} \Pr(\overline{s^a}, \overline{\sigma^a} \mid \Pr_0(S), \mathbf{A}, \mathbf{E})$. The root failure is guessed as s^a_0 but the normal state s_0. This is because we assumed that events are issued in the same order with the failure derivation.

Viterbi Algorithm [87] can solve $\bar{s} = \underset{\bar{s}}{\operatorname{argmax}} \Pr(\overline{s^a}, \overline{\sigma^a} \mid \Pr_0(S), \mathbf{A}, \mathbf{E})$. $\mathrm{Pmax}_{s^a_{t+1}}(t+1)$ is a maximum likelihood that Failure s^a_{t+1} occurs at t + 1 under $\overline{\sigma^a} = \sigma^a{}_0, \sigma^a{}_1, ..., \sigma^a{}_{t+1}$.

In Viterbi algorithm, the following equation is given.

$$\mathrm{Pmax}_{s^a_{t+1}}(t+1) = \mathbf{E}(s^a_{t+1}, \sigma^a_{t+1}) \max_k (\mathrm{Pmax}_{s^a_{t+1}}(t) \ \mathbf{A}(s_t, S_{t+1}))$$

$\mathrm{E}(s_i, \sigma_k)$ is a probability of Event σ_k under Failure s_i, which is Element (i, k) of Probability Matrix $\mathbf{E}$. $\mathrm{A}(s_i, s_j)$ is a probability of Failure s_j derives Failure s_i, which is an element (i, j)

of Probability Matrix **A**

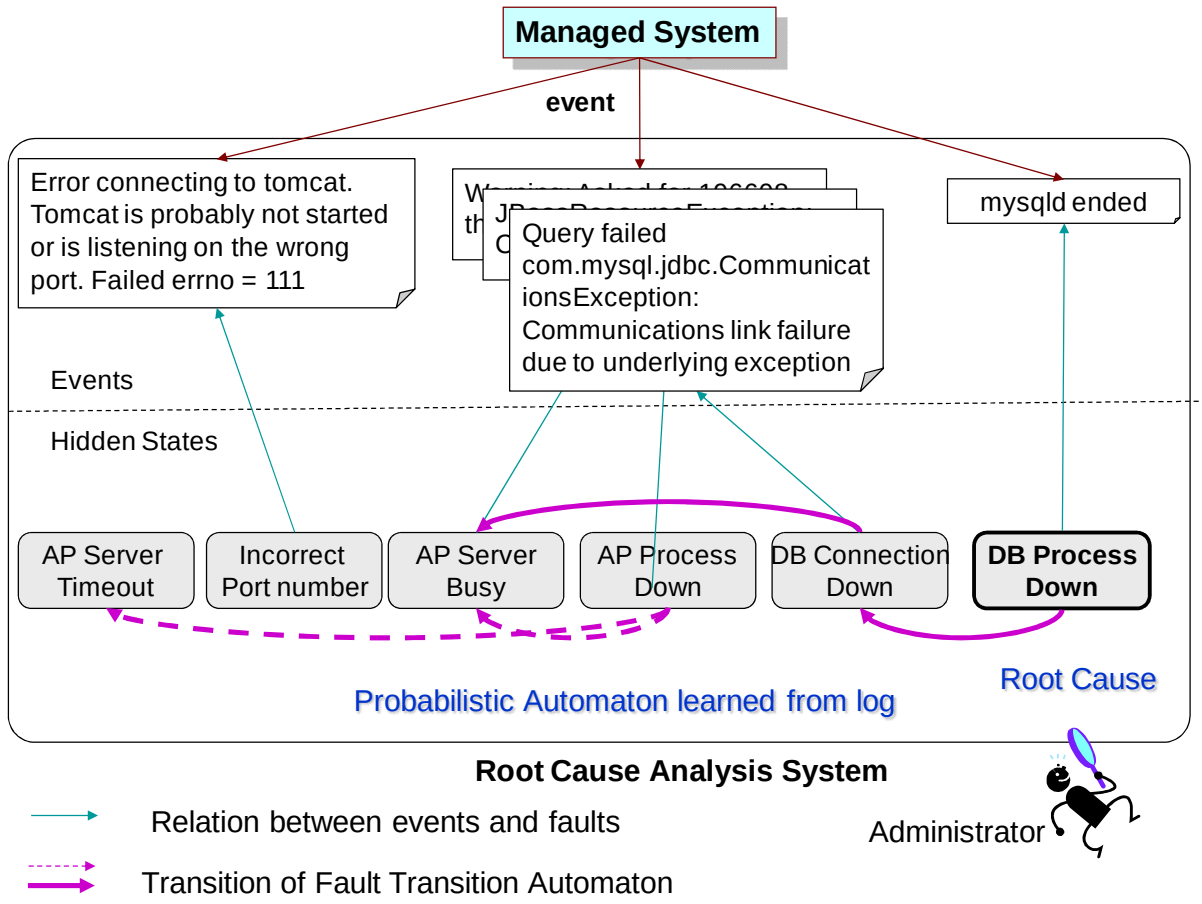

Figure 29: Failure derivation model with Hidden Markov Model

Computation times of the Baum-Welch algorithm and the Veterbi algorism are O(n), where n is the number of events for learning and for analysis. We, therefore, have to divide a long sequence of events into segments with suitable size. We assume that most of failure derivations occur in a short time. We, therefore, divide a sequence of events at the gap where time period between an event and the next event is longer than a given threshold. This is shown in upper part of Figure 30. The segment is called an "event region" in this thesis.

Several faults may occur in the same event region. We assume that, in a lot of cases, each of the faults issues a different sequence of events. We, therefore, can separate these

sequences in an event region by finding a low probability between the sequences. In some case, a sequence of faults $\overline{s^a} = s^a{}_0, s^a{}_1, ..., s^a{}_{n-1}$ found in this method may include several different sequences of faults caused by the faults. We regard as the sequence as different sequences if transition probability is low. For example in the lower part of Figure 30, we divide the sequence of failures at the line between Failure s_j and s_{j+1} if $Pr(s_{j+1} \mid s_j) < P_0$. P_0 is a given threshold called a divide probability in this thesis.

Event σ_k is not regarded to be issued from Failure s_j if $Pr(\sigma_k \mid s_i) < P_1$. It is regarded as errors of the proposed method. We can get sets of events from Event region R_i. Assume that h th set of events is $\Sigma_{i,h}$. A set of events of $\Sigma_{i,h} \cap R_i$ is events which failure derivation issues.

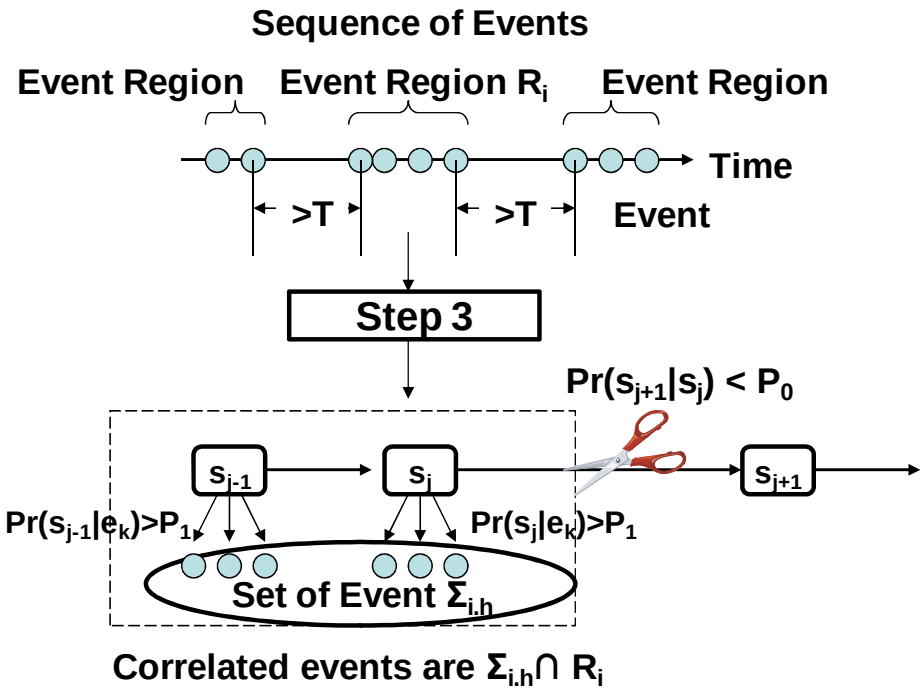

Figure 30: Event Region

5-5 Evaluations and discussion

Evaluation environment is a web three-tier system shown in Figure 31. The reasons why we adopt a web three-tiered system are following:

- It is commonly used.
- It is a typical distributed system, whose components are a web server, a set of application (AP) servers, and a DB server.
- Failures in a component can derive the failures in other components, and network failures may derive failures in application layers, and vise visa.
- We can use some different system configurations. For example, we can use WebLogic, WebOTX or some other servers as application servers. It means that we cannot define easily failure derivations.

We use MySQL as a DB server, Tomcat with JBoss as application servers, and Apache with mod_jk as a web server. A log collector collects events issued in the components

The event specifications used in this evaluation contains 223 lines, including 200 types of events, and 16 types of failures. We do not distinguish events of each application server from the events of application servers; we regard events of application servers as the same, whichever application server issues it.

We injected the following six kinds of failures into the evaluation system periodically.

- Termination of Httpd (i.e. web server process).
- Termination of JBoss (i.e. application server process)
- Termination of MySQL (i.e. DB server process)
- Termination of network in one of the application severs
- Termination of network in the web server.
- Termination of network in the DB server.

Evaluation indexes are the following:

- **Precision** is the ratio of correctly detected root failures in all injected faults. In short, (Precision) = (# of correctly detected root failures) / (# of injected faults) .
- **False positive rate** is the ratio of wrongly detected root failures in detected root failures. In short,
  (False positive rate) = 1 — (# of correctly detected root failures) / (# of detected root failures).

These two evaluation indexes are in trade-off relation. If we try to find all the root failures, we may detect wrong root failures. If we try to avoid from detecting wrong root failures, we tend to miss out correct root failures.

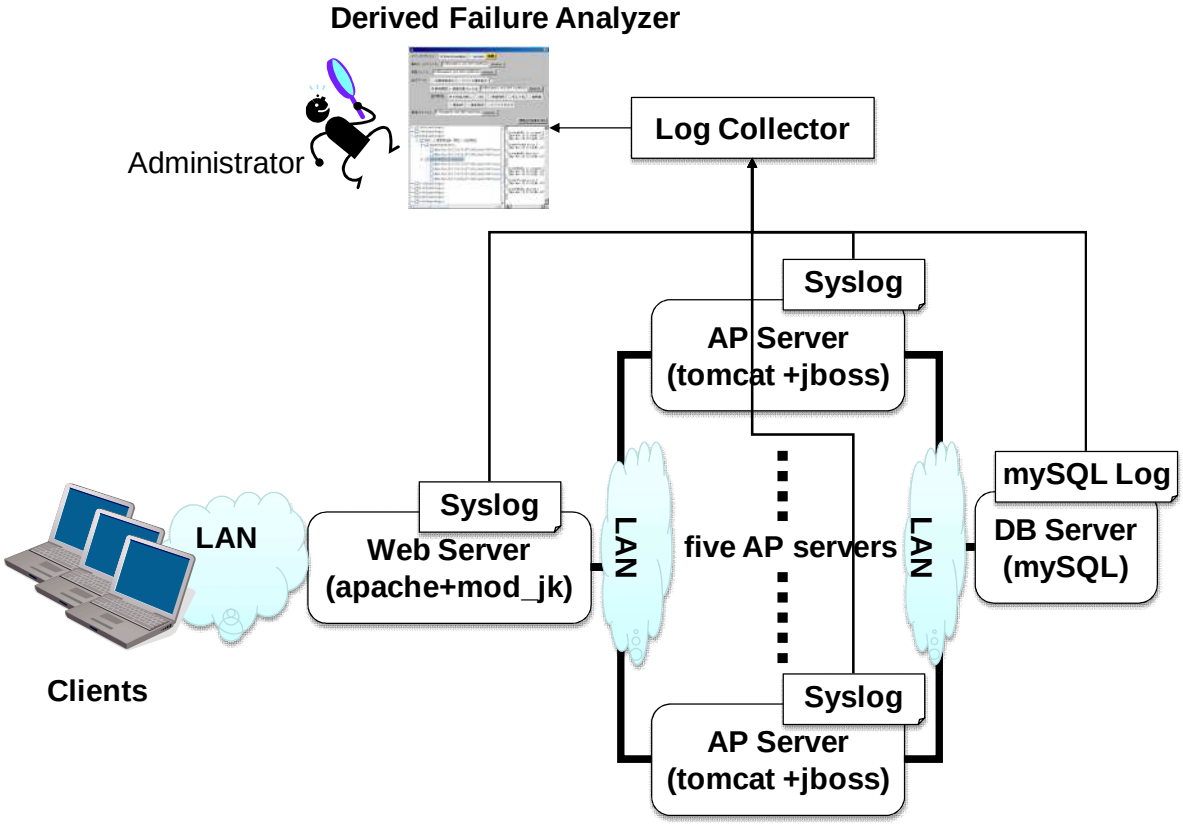

Figure 31: Evaluation Environment

The evaluation results are shown in Figure 32. The x-axis is a time period T in which no event occurs. It is used as index of separation of event regions. The size of an event region is generally long when T is large, while it is generally short when T is short. The y-axis shows precision and false positive rate. We find, in this evaluation, that the precision and the false positive rate tends to be low when T is large. In short:

- Precision is about 60% when T is short. However, false positive rate is about 70% because some event regions do not include the root failure as shown in Figure 33.
- False positive rate is almost 0% when T is long. However, the proposed method wrongly detects a root failure because two different sequences originated from two faults are regarded as one sequence. This is shown in Figure 34. We can adjust the divide probability P_0 for improvement of the precision.

  As shown in the evaluation, we can configure T and P_0 to adjust a desirable precision

and false positive rate. For example, we set T long to reduce the false positive rate. In this configuration, detected false derivation relation includes several root failures. However, an administrator can detect true root failures from the detected event derivation relation by his inspection. In other word, the proposed method helps the administrator by narrowing the root failures.

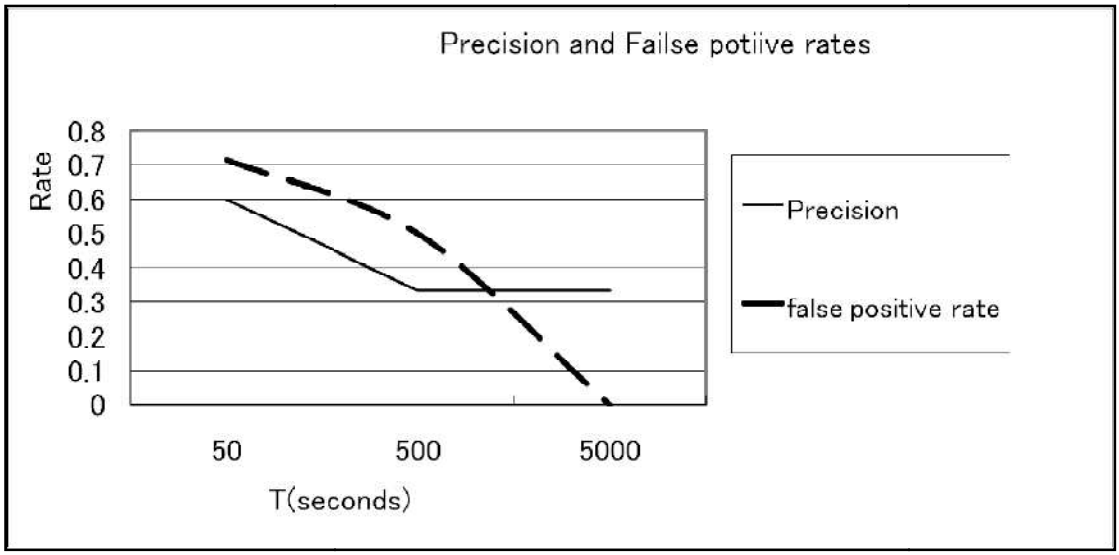

Figure 32: Precision and False positive rate

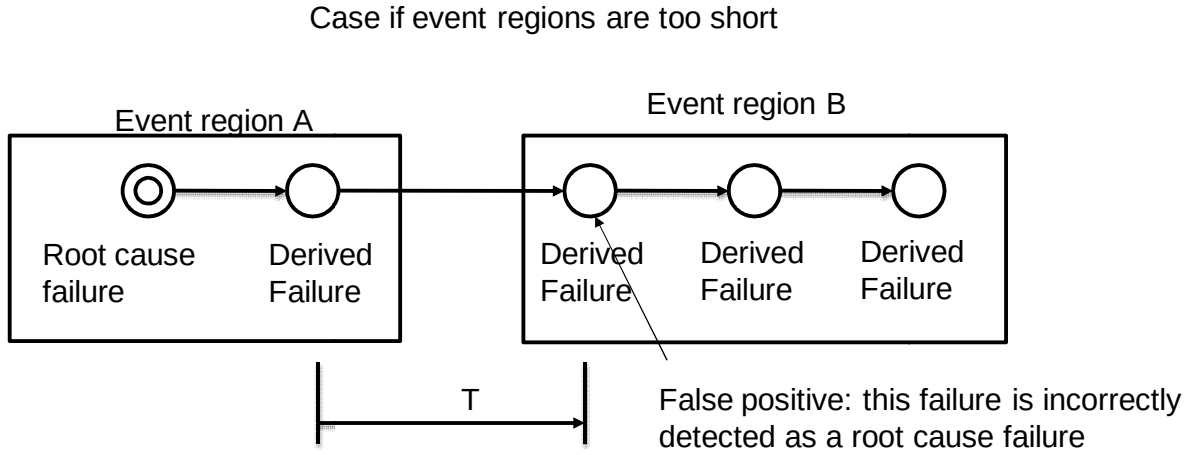

Figure 33: Case if event region is too short

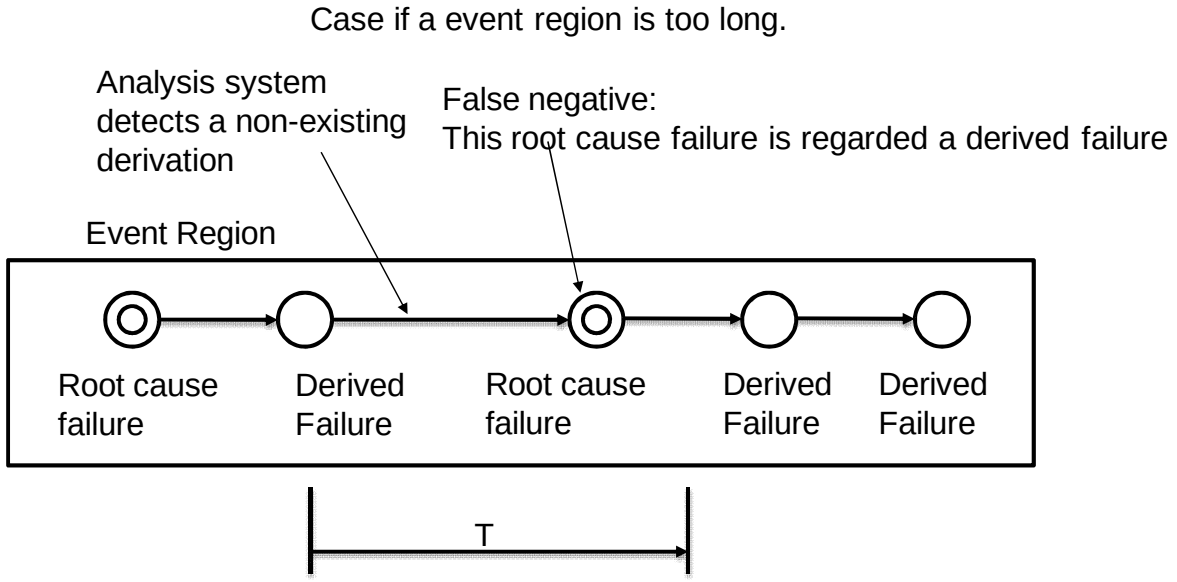

Figure 34: Case if event region is long

We, then, show, in Table 13, the top-five list of the probabilities of failure derivation generated in this evaluation. The derivation from one failure to the same failure is removed from this list because a failure repeatedly occurs until it is fixed.

Table 13 shows the derivation from "DB connection Error" to "DB connection Error" is 67%. It is a typical example that the method can model the derivation from a failure in network connection to an application failure.

Table 13 also shows that the derivation from "DB query Error" to "Servlet Error" is 50%. It is a typical example that the method can model the derivation between application failures.

Table 13: Top five probabilities of failure derivation

Source Failure	Derived Failure	Probability
Servlet Error	DB connection Error	100%
httpd starts	DB connection Error	100%
DB connection Error	DB query error	67%
DB query error	Servlet Error	50%
httpd ended	httpd starts	33%

5-6 Conclusion of low cost operation for high-dependable systems

Dependencies among applications and between applications and networks become complex as systems provide higher functions. Failure derivations become complex and it is hard to detect a root failure. In this chapter, we propose a method which can model failure derivations in the managed system with likelihood analysis. It learns a failure derivation model from the existing event log. We show that this method is effective through the empirical evaluation.

Machine learning method depends on the log for learning. Therefore, if the configuration of a managed system is changed, the failure derivation analyzing system has to learn a log for the managed system with the new configuration, again. We think that the SI-er of the managed system always tests the system whenever the configuration is changed, and a new log can be generated in the test. We can use the new log for learning the derivation relation.

Another approach to the issue of the configuration change is an on-line machine learning approach. It can re-learn the failure derivation relation while it manages the new managed systems. It is future work.

We know that this method is difficult to adapt to a large distributed system with a large number of components. For example, we define a failure in the event specifications as a pair of a failure and a device issuing the event. This results in a lot of definitions of failures if a lot of devices exist in the managed system. Assume that the number of devices is N, and an average number of failure types is n, then the number of total failures in the event specifications is $|S| = Nn$. It is known that calculation times of Baum-Welch algorithm and Viterbi algorithm are $O(|S|^2)=O(N^2n^2)$ where S is a set of failures. It is future work.

Chapter 6
Conclusion

Chapter 6. Conclusion

It is said that the management and operation costs occupy a large part of the Total Cost of Ownership (TCO) of ICT systems. The management and operation costs consist of the capital investment cost of operation systems and the operation cost of the ICT system. We insist that these two costs must be reduced for the reduction of the management and operation costs.

Firstly, in this thesis, we proposed an efficient management agent platform, which can efficiently run even on an inexpensive and poor environment. We also proposed a method of easily developing an agent application on the platform. These two are ones of solutions to reduce the capital investment cost of operation systems.

Secondly, we proposed a fault analysis method in order to reduce the operation cost of the ICT systems. Administrators always watch whether managed systems run correctly. If they find failures on the systems, they have to quickly analyze the cause of the failures. The analysis requires a lot of knowledge of the managed ICT systems, and it is a tough job. The proposed fault analysis method can reduces a burden of the administrators, and it results in the reduction of personnel costs used in the daily operations.

We mention contributions of this thesis in the following.

Firstly, we developed a Q3 management agent platform embedded in network elements. The standardized specifications of the Q3 agents have rich functionalities. They require a lot of hardware resources, such as much capacity of memory. Cost reduction of management agents is important because huge number of management agents embedded in network elements placed in access networks. We revise the structure of the containment tree and other data structures stored in a Q3 agent, and we successfully run a Q3 agent on the PowerPC board with 32MB memory.

Secondly, the capital investment of management agents includes the development cost of a Q3 agent application program as well as the cost of hardware where the Q3 agent runs. We proposed a DSL approach to the reduction of the development cost. We point out one of the important classifications of maintenance, which is defined as an affected maintenance. We showed that it occupies 8% of total maintenances in an existing project. In order to reduce the cost of the affected maintenance, we developed a DSL toolkit called Rosetta. DSLs developed with Rosetta are easily maintained. We show that Rosetta can reduce the affected maintenance cost more than the visitor design pattern can.

The Q3 agent platform and Q3 agent application programs developed with Rosetta are

actually used in network elements in base stations of intercontinental telecommunication cables.

Finally, we proposed a fault analysis method for reduction of the operation cost in the fault management. The propose method can detect a root cause from a lot of events issued by derivative failures. There are many works of fault analysis methods for managed systems with fixed structure. However, there appear a lot of distributed systems in these days. The configurations of the distributed systems are flexible, and the relations of failure derivation also become changeable. In addition, failure derivation occurred in the application program of the managed systems is implicit to the administrators, and it is difficult to find the root cause of failures. Our method learns a failure derivation model from an existing event log, and it detects a root cause by using the model. The exiting log is an outcome of the configuration of managed systems and failure derivation relation in the systems including the application program. We show, in our experiments, that the method can correctly learn the model, and the proposed method is effective if the parameters suitable to administrators' policy are given.

The cost structure shown in Figure 3 is not changed, but the portions of these costs are changed. The hardware cost for management agents is getting small because the prices of hardware including memory are getting quickly low. Therefore, we should struggle with the other two costs.

We will especially focust on the operation cost. As application software is getting more complex, we expected that faults in application software will become more complex. It will become one of main problems in fault analysis. We think that failure detection in application software is a future work. A lot of application program written in Java, VB, and so on. These byte-code programs are easy to be inspected. We can grasp the situation and the structure of the application software in comparison with C/C++ binary codes. It may be potential to new researches.

Bibliography

[1] Implementation of a Fast Q3 Agent Platform for Agents Embedded in Network Elements, IPSJ Transactions Vol.41 No.4 pp.1226-1233, May. 2000. (in Japanese)

[2] Toshio TONOUCHI, Takashi FUKUSHIMA, Asuka MANKI, Shin NAKAJIM, An Implementation of OSI Management Q3 Agent Platform for Subscriber Networks, Int Conf on Communication (ICC), pp. 889-893, 1997

[3] Toshio TONOUCHI and Shin NAKAJIMA, “A Main-memory MIB Platform for Fast and Compact OSI Management Agents”, The 52th National Convention of IPSJ, Vol52, No.1 pp.99-100, Mar. 1996. (In Japanese)

[4] Toshio TONOUCHI and Shin NAKAJIMA, “Architecture of OSI MIB Platform and its Performance Evaluation” , JSSST Annual Conference, Sep. 1996.

[5] Toshio TONOUCHI, Masahiro TAKEI, Shin NAKAJIMA, and Shouichiro NAKAI, “A Memory Management Architecture of MIB for OSI System Management”, IEICE Society Conference No.2 , p.116, Sep. 1995. (In Japanese)

[6] RFC-1151, A Simple Network Management Protocol (SNMP), 1987

[7] ISO/IEC-9596-1, Common Management Information Protocol, 1991

[8] Toshio TONOUCHI, and Shin NAKAJIMA, “A Toolkit for Developing DSL Translator”, IPSJ Transactions Vol.43 No.1 pp.146-155, Jan. 2002. (in Japanese)

[9] Toshio TONOUCHI and Shin NAKAJIMA, “A Template Driven method for Developing DSL Translators and Implementation of a Translator Toolkit based on the Method”, JSSST Annual Conference, Sep. 1999. (In Japanese)

[10] Toshio TONOUCHI and Shin NAKAJIMA, “A Re-targetable GDMO Translator”, JSSST Annual Conference, Sep. 1997.

[11] Toshio TONOUCHI and Shin NAKAJIMA, “Implementation of GDMO Translator by Using Language Toolkit Rosetta”, Technical Reports of IEICE(TM1999-125) Vol.99, No.101 pp. 63-70, 1999/6/4

[12] ITU-T Recommendation X.722, “Guidelines for the Definition of Managed Object”, 1992

[13] Toshio TONOUCHI and Masayuki MURATA, “Root cause analysis technique for

derivative failures with implicit dependencies", accepted in IEICE Transactions on Communications, Vol.J92-B, No. 8, Aug. 2009. (in Japanese)

[14] Toshio TONOUCHI, "Fault Analysis under the Changeable Environment", Technical Reports of IEICE ICM2008-33, pp.55-60, 2008/7

[15] Toshio TONOUCHI, "Fault analysis for derived failures with maximum likelihood", Technical Reports of IEICE (TM2007-39) pp.29-34, Nov. 2007.

[16] RFC-1213, Management Information Base for Network Management of TCP/IP-based Internets: MIB-II

[17] ITU-T, "M.3050: Enhanced Telecom Operations Map (eTOM) The business process framework", 2004/6

[18] Common Information Model (CIM) Specification, 2.2, June 14, 1999 - Downloadable from http://www.dmtf.org/spec/cims.html

[19] DSL Forum, "TR-069 Amendment 1-CPE WAN Management Protocol", Dec., 2006

[20] Toshio TONOUCHI, Norihito FUJITA, Tomohiro IGAKURA, Naoto MAEDA, Yoshiaki KIRIHA, "Integrated Service Navigation Framework for Ubiquitous Networking", NEC Journal of Advanced Technology (JAT), Sep. 2004

[21] Hua SI, Yoshihiro Kawahara, Tomohiro Igakura, Toshio Tonouchi, Hiroyuki Morikawa and Tomonori Aoyama, "A Hybrid Context-aware Service Platform Based on Stochastic and Rule-Description Approaches", Annual International Conf on Mobile & Ubiquitous Systems: Networks & Services (MOBIQUITOUS) Demo session, Jul. 2006

[22] Tomohiro Igakura, Toshio Tonouchi, Yoshihiro Kawahara, and Hua SI, "Transition of "Prohibition Policies" for a Context-aware Service Recommendation System", IEICE (TM), 2005 (In Japanese)

[23] Yoshihiro Kawahara, Hua SI, Tomohiro Igakura, Toshio Tonouchi, Hiroyuki Morikawa and Tomonori Aoyama, "An Information Appliance Control Mechanism based on User Action History and Constraint Conditions", IPSJ SIG UBI technical reports Vol.2006, No.14(20060216) pp. 55-60

[24] Yoshihiro Kawahara, Hua SI, Tomohiro Igakura, Toshio Tonouchi, Hiroyuki Morikawa and Tomonori Aoyama, "Design of a Context-aware Service Recommendation System Based on Action History and Constraints", Proceedings of the Society Conference of IEICE Vol.2005(B), No.2(20050907) p. 446, Sep. 2005. (In Japanese)

[25] Toshio TONOUCHI, "Implementation of Dynamic and Virtual Meeting Space with

Server/Network Integrated Provisioning", Technical Reports of IEICE (TM2004-24), pp.19-24, Jul. 2004.(In Japanese)

[26] Toshio TONOUCHI, Yasuyuki BEPPU, Management of Autonomic Scalable Load-balancing for Ubiquitous Networks, APNOMS 2005, LNCS 4238 pp. 33-42

[27] K. Yata, I.Yoda, K. Minato, N. Fujii, "ATM transport network operation system based on object oriented technologies", GLOBECOM 94, pp.838 - 842 vol.2, Dec 1994

[28] RFC 1189, "The Common Management Information Services and Protocols for the Internet (CMOT and CMIP)", 1990

[29] S.H. Chae, et. al, "Implementation of GDMO to IDL Translator and CORBA/CMIP Gateway for TMN/CORBA Integration", APNOMS 1998

[30] Christy Coffey, et al, "Delivering a Multi-Vendor End-to-End Service Management Infrastructure to the UK Ministry of Defense using NGOSS", TM Forum, January 11, 2005

[31] Sloman, M. S., and Moffett, J. D.: "Domain Management for Distributed Systems", IFIP Symp. on Integrated Network. Management, North Holland, pp 505–516, 1989

[32] Damianou, N., Dulay, N., Lupu, E., Sloman, M., Tonouchi, T, "Tools for Domain-based Policy Management of Distributed Systems", IFIP/IEEE Network Operations & Management Symposium (NOMS) 2000

[33] Naoto MAEDA, Tomohiro IGAKURA, and Toshio TONOUCHI, "A Check Technique for Policy-based Management Systems", JSSST 23th Annual Conference, Sep. 2006. (In Japanese)

[34] Naoto MAEDA and Toshio TONOUCHI, "Safe and Efficient Policy Processing Framework for Distributed Systems Management", JSSST 21th Annual Conference, Sep. 2004 (In Japanese)

[35] Naoto MAEDA, Tomohiro IGAKURA, and Toshio TONOUCHI, "A Dependable Framework for Policy-based Management Systems", Technical Report of IEICE (TM2005-73) pp. 103-108, Mar. 2006 (In Japanese)

[36] Naoto MAEDA and Toshio TONOUCHI, "Error-Prone Policy Analysis for Distributed Systems Management", Technical report of IEICE (TM2003-122), Vol.103, No.701 pp. 67-72, 2004/3/5 (In Japanese)

[37] Naoto Maeda and Toshio Tonouchi, "An Analysis Method for the Improvement of Reliability and Performance in Policy-Based Management Systems", DSOM 2004,

LNCS 3278 pp. 88-99, 2004

[38] Tomohiro IGAKURA and Toshio TONOUCHI, “Policy management architecture for coordination between policy definition and system monitoring”, in Proceedings of the IEICE General Conference Vol.2005 No. 2 pp.”S-71”, Mar. 2005. (In Japanese)

[39] Tomohiro IGAKURA and Toshio TONOUCHI, “A Policy Transition Model to realize policy control depending on system states”, in Proceedings of the IEICE General Conference Vol.2004 No. 2 pp.607, Mar. 2004. (In Japanese)

[40] Tomohiro IGAKURA and Toshio TONOUCHI, “Policy-Based Service Management with State Transition Model”, Technical report of IEICE (TM2003-22) Vol.103, No.180 pp. 19-24, 2003/7/4 (In Japanese)

[41] Tomohiro IGAKURA, Toshio TONOUCHI and Yoshiaki KIRIHA, “Policy-based Interaction Management for Application Services”, in Proceedings of the IEICE General Conference Vol.2003(B) No.2 p. 674, 2003/3/3

[42] Tomohiro IGAKURA, Toshio TONOUCHI and Yoshiaki KIRIHA, “Policy-based Application-level Message Routing”, Technical report of IEICE (TM) Vol.102, No.462 pp. 9-14, 2002/11/14 (In Japanese)

[43] Tomomi YOSHIOKA, Tomohiro IGAKURA and Toshio TONOUCHI, “Consistent Configuration of Network Elements among Domains with Policy Server”, Technical report of IEICE (TM2003-9) Vol.103, No.43 pp. 7-12, 2003/5/9 (In Japanese)

[44] Toshio TONOUCHI, “A Mathematical approach to definition and fulfillment of requirements for development of a policy management tool”, Technical Report of IPSJ (2001-SE-135-1) Vol.2001, No.114 pp. 1-8, 2001/11/21 (In Japanese)

[45] Toshio TONOUCHI, “Class of Service with an Access-control Policy System”, IPSJ SE Object-oriented Symposium (OO), pp. 51-58 2002

[46] RFC 2748, “The COPS (Common Open Policy Service) Protocol”, January 2001

[47] IBM, “An architectural blueprint for autonomic computing”, 2004

[48] NGMN Alliance, “Use Cases related to Self Organizing Network.Overall Description”, 2007/4/17

[49] Shinji NAKADAI and Toshio TONOUCHI, “Case-based Fault Detection using Support Vector Machine”, Technical report of IEICE (ICM), Nov. 2008. (In Japanese)

[50] Tomohiro IGAKURA and Toshio TONOUCHI, “Construction and evaluation of rule-based fault management system”, Technical Report of IEICE (ICM2008-5) pp.23-28,

May 2008. (In Japanese)

[51] Malgorzata Steinder, and Adarshpal S. Sethi. “A survey of fault localization techniques in computer networks”. Elsevier Science of Computer Programming Journal, pp. 165-194, 2004.

[52] L. Lewis. *A case-based reasoning approach to the resolution of faults in communications networks*. In IM'93 [19], pp. 671--681.

[53] IBM “Automate data collection for problem determination, Part 4: The Automated Problem Determination Tool” http://www-128.ibm.com/developerworks/autonomic/library/ac-autopd4.html

[54] Y. Watanabe, Y. Matsunaga, K. Kobayashi, T. Tonouchi, T. Igakura, and S. Nakadai, “UTRAN O&M Support System with Statistical Fault Identification and Customizable Rule Sets”, NOMS 2008, pp. 560-573

[55] K. Sun et. al. “A State Machine Approach for Problem Detection in Large-scale Distributed Systems”, NOMS 2008

[56] Bahl et. al., “Towards Highly Reliable Enterprise Network Services Via Inference of Multi-level Dependencies”, SIGCOMM 2007

[57] R. Kompella et. al., “Detection and Localization of Network Black Holes”, INFOCOM2007

[58] ITU X.710 ISO/IEC 9595, Common Management Information Service Definition

[59] ITU X.208 ISO/IEC 8824, Specification of Abstract Syntax Notation One (ASN.1)

[60] Masahiro TAKEI, Toshio TONOUCHI, Shouichiro NAKAI, and Shin NAKAJIMA, “Communication Protocol between MIB and Network Element”, IEICE Society Conference, 1995

[61] Rose, T.: “The ISO Development Environment: User's Manual”.

[62] RFC 1006, “OSI transport services on top of TCP”, 1991.

[63] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Pattern: Element Reusable Object-oriented Software", Addison-Wesley, 1994

[64] S. Sasaki et al., ``Management Platform for TMN Agent System Using Real-time Operation System'' (in Japanese), TECHNICAL REPORT OF IEICE, IE94-120, 1994

[65] G. Pavlow et al., ``The OSIMIS Platform : Making OSI Management Simple,'' Proc. of the 4th Integrated Network Management, pp. 480--493, 1995

[66] ITU-T, M.3100, Generic Network Information Model, 1995

[67] DSET Corp.: Agenet ToolKit, http://www.dset.com/tmn/agent_toolkit1.html, 1998

[68] SUN Micro Systems: Solstice TMN Toolkit, http://www.sun.co.jp/software/products/ENP/telecom/TMNtoolkit.html, 1999

[69] J. Haritsa et al., ``Design of the MANDATE MIB," Proc. of 2nd Integrated Network Management, pp.85--96, 1993

[70] Nishiyara Satoshi, Yokota Hidetoshi, Obana Sadao, Suzuki Kenji, “Hash-based High-Performance Name Resolution Method for OSI Directory Information Base (DIB)”, Vol. 35, No.12, pp.2762-2773, 1994

[71] Nakai, S. “MIB Design for Network Management Transaction Processing”, Integrated Network Management III, pp. 97-108, 1993

[72] Toshio TONOUCHI, Ken NAKAYAMA, Satoshi MATSUOKA, and Satoru KAWAI, “Creating Visual Objects by Direct Manipulation”, in Proceedings of IEEE Visual Language (VL) 92, pp.92-101

[73] Richard B. Kieburtz and et. al., “A Software Engineering Experiment in Software Component Generation", In Proceedings of ICSE, pp. 542 552, 1996

[74] Fjeldstad and Hamlen, “Application Program Maintenance Study-Report to Our Re-spondents", IBM Corporation, DB Marketing Group, 1979.

[75] Carma McClure, “Software Reuse Techniques: Adding Reuse to the System Development Process ", ISBN 978-0136610007, 1993

[76] David L. Atkins, Tomas Ball, Glenn Bruns, and Kenneth Cox, “Mawl: A Domain-Specific Language for Form-Based Services", IEEE Transactions on Software Engineering, Vol. 25, No. 3, pp. 334 346, May/June. 1999

[77] Arie van Deursen and Paul Klint, “Little Languages: Little Maintenance?", In Procedings of First ACM SIGPLAN Workshop on Domain-Specific Languages, DSL '97, pp. 109 127, 1997

[78] J. C. Cleaveland, “Building Application Generators", IEEE Software, pp. 25 33, July. 1988

[79] James M. Neighbors, “The Draco Approach to Constructing Software from Reusable Components", IEEE Transactions on Software Engineering, Vol. 10, No. 5, pp. 564 574, Sep. 1984

[80] Peter Freeman, “A Conceptual Analysis of the Draco Approach to Constructing Software Systems", IEEE Transactions on Software Engineering, Vol. 13, No. 7, pp. 830 844,

July.1987

[81] Richard B. Kieburtz and et. al., “A Software Engineering Experiment in Software Component Generation", In Proceedings of ICSE, pp. 542 552, 1996

[82] Scott Hudson, “CUP Parser Generator for Java", http://www.cs.princeton.edu /~appel/modern/java/CUP

[83] Megginson Technologies, "SAX 2.0: The Simple API for XML", http://www.megginson.com/SAX/index.html, Jun. 2000.

[84] G. Jakobson and M. D. Weissman,, “Alarm Correlation”, IEEE Network, pp. 52-59, Nov. 1993

[85] K.-W.E. Lor, ``A network diagnostic expert system for Acculink multiplexers based on a general network diagnostic scheme", in: H.G. Hegering, Y. Yemini (Eds.),

[86] Jinsik Kim, Youngmoon Yang, Sungwoo Lee, Sukji Park, Byungdeok Chung, ``Fault Localization for Heterogeneous Networks using Alarm Correlation on Consolidated Inventory Database," APNOMS 2008, pp.82-91 LNCS 5297

[87] L. R. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition," Proceedings of the IEEE, vol. 77, pp. 257-286, 1989.

[88] D. Patterson , et. al., “Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies”, UC Berkeley Computer Science Technical Report UCB//CSD-021175, March 15, 2002

[89] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, Eric Brewer, ``Pinpoint: Problem Determination in Large, Dynamic Internet Services", DSN02, 2002

[90] Shaula Alexander Yemini, Shmuel Kliger, and Eyal Mozes, Yechiam Yemini and David Ohsie, *High Speed and Robust Event Correlation*, IEEE Communications Magazine, May 1996

[91] Y Matsunaga, and Kenji Yamanishi ， ” An Information-theoretic Approach to Detecting Anomalous Behaviors”, FIT 2003, pp 123-124, 2003 (In Japanese)

[92] H. Chen and K. Yoshihira, “Modeling and tracking of transaction flow dynamics for fault detection in complex systems,” IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 4, pp. 312–326, 2006.

[93] ANA, News release, 2008/9/18, http://www.ana.co.jp/topics/notice080914/index.html

[94] Impress, “DoCoMo lost billing data of i-mode” (Title is translated by Tonouchi), 2005/04/13

http://k-tai.impress.co.jp/cda/article/news_toppage/23452.html

[95] Nikkei ITPro, “NTT-W explained Cause of IP Phone Accident” (Title is translated by Tonouchi), 2006/04/25
http://itpro.nikkeibp.co.jp/article/NEWS/20060425/236243/

[96] Nikkei ITPro, “Software bug eject travelers from Airport” (Title is translated by Tonouchi), 2008/3/15. (In Japanese)
http://itpro.nikkeibp.co.jp/article/COLUMN/20060602/239857/

[97] Nikkei ITPro, “System down by Software bug in Tokyo Stock Exchange Market”, 2005/11/18. (In Japanese)
http://itpro.nikkeibp.co.jp/article/COLUMN/20051111/224404/

[98] Wikipedia, “Case of Miss-order to J-COM stocks”. (In Japanese)
http://ja.wikipedia.org/wiki/%E3%82%B8%E3%82%A7%E3%82%A4%E3%82%B3%E3%83%A0%E6%A0%AA%E5%A4%A7%E9%87%8F%E8%AA%A4%E7%99%BA%E6%B3%A8%E4%BA%8B%E4%BB%B6

[99] H. Watanabe, “Resent system accidents and insurance for the accidents” (Title is translated by Tonouchi) , September 2006. (In Japanese),
http://www.csaj.jp/info/06/20060912_us_it_softmarket.pdf

[100] @IT, “Executives explained careless mistake caused the system down in JASDAC” (Title is translated by Tonouchi), 2005/8/30. (In Japanese),
http://www.atmarkit.co.jp/news/200508/30/jasdaq.html

[101] Nikkei, ”ATM Troble in Risona Bank” (Title is translated by Tonouchi), in Nikkei Computer 2005/05/30. (In Japanese)

Appendix A Trouble cases

Table 14: Accidents of ICT Systems

#	Trouble	Detail	Date	Environmental factors	Internal failure derivation	Fault	Failures	Error event	Difficulties for fault analysis
1	ANA Boarding System [93]	ANA Boarding System could not accept a check-in operation. The cause is in the encryption module with license. The lease fell down, and the system can use the module	2008/9/6	Out of lease period	Inaccessible to encryption module	Configuration of license period	Termination of function	レ	implicit failure derivation
13	Billing leak in NTT [94]	Billing system in NTT was miss-configured as one of two jobs asleep, and it only count in half of billing data.	2006	N/A	miss-configuration	miss-configuration	malfunction	---	silent failure
14	Disconnection NTT IP Telephone [95]	IP telephones were not available because SIP servers cannot keep connections to broken servers, and it cannot afford phone connections.	Mar 31, April 13 and 21, 2006	N/A	Failure in servers other than SIP	Software bug	malfunction	---	implicit failure derivation
23	Trouble Baggage monitoring system in airport. [96]	Airport clerks run the baggage monitoring system in test mode. They misunderstand the message in the test mode that system may be down. They check all baggage by hand	2006/4/1	N/A	Miss-operation	N/A	Miss-operation	---	Misunderstanding to the operation

24	System in Tokyo Stock Exchange Market did not re-start. [97]	Some operations for restart is missing in operation manual. So the system did not restart.	2005/11/1	N/A	N/A	Incomplete manual	malfunction	---	Finding Incompletion of manual is difficult for everyone but manual writer.
25	Case of Miss-order to J-COM stocks [98]	In specification, the system alert a user if abnormal price. But the mechanism is not implemented in the system.	2005/12/8	N/A	Miss-operation	Lack of Fool proof	Miss-operation	レ	Nothing
26	Failure in auto pilot system in Boeing 777 [99]	Fight 124 (Boeing 777) from Australia to Malaysia suddenly went up and down The software bug in sensors monitoring flight height and its speed.	2005/3/1	N/A	N/A	Software bug	Miss-operation	---	Emerge case
27	Termination of X-ray baggage monitoring system Nashville airport [99]	Software update including bug stops the X-ray system. Airport clerk checked baggage by hands.	2004	N/A	N/A	Software bug	termination	レ	Fault localization
28	Troubles in Prius [99]	Software bug in ECU program cause engine trouble when	2005/6/1	on high-spe	N/A	Software bug	malfunction	---	Emergency case

		Prius run in high-speed		ed driving					
29	Voting counting machine on President Election in 2004, US[99]	Voting counter overflows its counter when the number of votes is over a threshold.	President election in 2004, US	Overflow	N/A	Software bug	Miss-operation	---	Fault localization
30	Voting counting machine on President Election in 2004, US [99]	When votes data are loaded into the voting counter again, it twice counted the votes		N/A	N/A	Software bug	malfunction	---	Fault localization
31	System downin JASDAC [100]	Bugy system e estimates the max connection of submodules too small. It causse mis-configuration. Test phase of the system, only small number of connection only checked. A large number of connections is no tested.	2005/8/29	N/A	mis-configuration	N/A	malfunction	レ	Fault localization
32	ATM troubles in Risona Bank. [101]	Cash cards the merged bank cannot be used in ATM united new bank. Software bug is a cause.	2005/5/6	N/A	N/A	Software bug	malfunction	---	Fault localization